Universidad de Alcalá
Escuela Politécnica Superior

Máster Universitario en Ingeniería de Telecomunicación

Trabajo Fin de Máster

Anomalous Behaviour Detection in Video Surveillance Scenes

Autor: Marcos Baptista Ríos

Director: Cristina Losada Gutiérrez

2017

UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNIA SUPERIOR

Máster Universitario en Ingeniería de Telecomunicación

Trabajo Fin de Máster

Anomalous Behaviour Detection in Video Surveillance Scenes

Autor: Marcos Baptista Ríos

Director: Cristina Losada Gutiérrez

Tribunal:

Presidente: Marta Marrón Romera

Vocal 1º: Roberto López Sastre

Vocal 2º: Cristina Losada Gutiérrez

Calificación: .....

Fecha: .....

Como todo, a mis padres..…

"Nunca puedes impedir que haga cosas, porque si no, no le pasaría nada."

Dory, Buscando a Nemo

Agradecimientos

Después de unos cuantos años, son algunas las personas que en mayor o menor medida me han ayudado a llegar hasta aquí.

En priner lugar, tengo que agradecer a GEINTRA, más en concreto a Marta y Cristina, la confianza que, hace ya algunos años, pusieron en mí. No cabe duda que de no haberme ofrecido la oportunidad de trabajar con ellas, las cosas habrían salido de otra manera completamente distinta. Gracias.

También quiero acordarme de toda la gente del *David's People*. Magníficas personas que siempre están dispuestas a dejar lo que estén haciendo para ayudarte. Mucho de lo que sé es gracias a ellos. Sobre todo a los de siempre: *El Jefe* David Jiménez, José, Casillas, David *Becario* y Juan. Gracias amigos. "Es el ciclo sin fin..."

No me puedo olvidar de los compis de clase que se convirtieron en amigos. En espcial María y Raquel por hacerme saber siempre que están ahí. Gracias. "Este es el definitivo máximo". "Fin del bucle, tía!!"

Y por último, todo lo que he conseguido se lo debo a mi madre y a mi padre. Nunca nada de lo que diga o haga podrá ser suficiente para agradecer toda la dedicación que habeís puesto en mí. "No tengo ni idea de lo que me estás contando, pero a mí me parece que está bien!!"

Resumen

El presente Trabajo Fin de Máster consiste en la implementación y evaluación de un método de reconocimiento de acciones adecuado para ser utilizado en un sistema de detección de anomalías. Dicho método es el propuesto por el trabajo titulado *Long-term Recurrent Convolutional Networks for Visual Recognition and Description*, el cual está compuesto por dos fases basadas en CNN y LSTM. Esta configuración se entrena en conjunto y es capaz de aprender dependencias temporales entre las características visuales extraídas.

Los resultados obtenidos sobre los conjuntos de evaluación de la base de datos UCF101, muestran que este método supera a otros modelos que no tienen en cuenta las relaciones temporales en su estructura. Además, el rendimiento que ofrece es comparable al de otros métodos del estado del arte.

Palabras clave: Reconocimiento de acciones, detección de anomalías, CNN, LSTM, dependencias temporales, características visuales.

Abstract

The master's thesis herein presented consists of implementing and evaluating a state-of-the-art action recognition method suitable for an anomaly detection system. The method is the one proposed in *Long-term Recurrent Convolutional Networks for Visual Recognition and Description*, which is a two-stage configuration based on Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM), end-to-end trainable and capable of learning time dependencies between visual features.

The results obtained over the test splits of the UCF101 dataset show that this method outperforms other models that do not learn temporal dynamics. In addition, the performance of the LRCN is comparable to that of other state-of-the-art methods.

Keywords: Action recognition, anomaly detection, CNN, LSTM, time dependencies, visual features.

Extended Abstract

Video surveillance can be defined as the process of monitoring or observing the activities, behaviour or movements of an individual or a group to provide enhanced security. This task is an obvious candidate to become automated as it is very monotonous and boring for the security staff who performs it.

The goal of the video surveillance task is to detect if something occurs without following the expected behaviour. In other words, the task is about detecting anomalies. In the human behaviour context, an anomaly would appear when an individual performs an action that does not correspond to the usual behaviour, therefore a possible anomaly detection system could include a human activity recognition method.

The master's thesis herein presented consists of implementing an evaluating a state-of-the-art action recognition method suitable for an anomaly detection system. The method is the Long-term Recurrent Convolutional Network (LRCN) shown in figure 1 and proposed by [1], which is a two-stage configuration based on Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM).

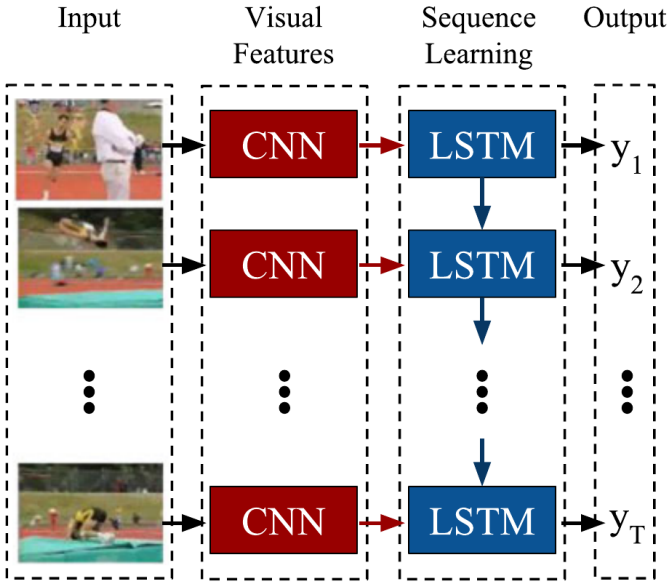


Figure 1: Long-term Recurrent Convolutional Network architecture [1].

Prior to the analysis of the developed method, a study of the recent literature on anomaly detection has been made so as to determine the route the scientific community is following. This study suggests that, although there are some recent works with semi-supervised anomaly detection systems, the unsupervised ones are the most popular. Furthermore, in spite of the fact that trajectory-based and low-level feature methods have traditionally been used, a new line of research based on deep learning architectures is beginning to be widely explored.

Regarding the implementation of the LRCN, the visual feature extraction stage utilises the 6 layer CNN model illustrated in figure 2, while the sequence learning stage works with LSTM networks as can be seen in figure 3. Due to this structure, the time-relationship between the visual features can be modelled.

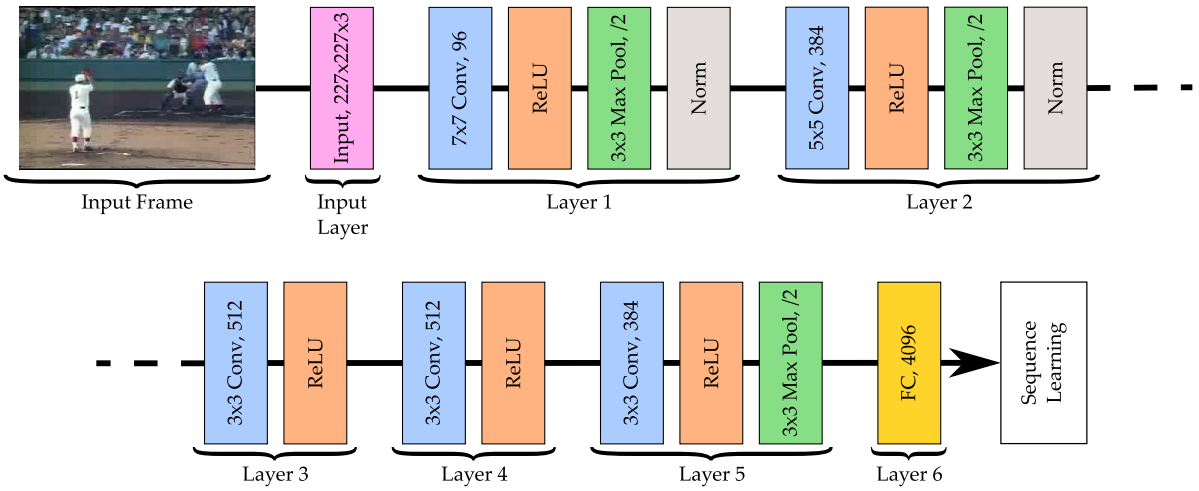


Figure 2: CNN used in the visual feature extraction stage.

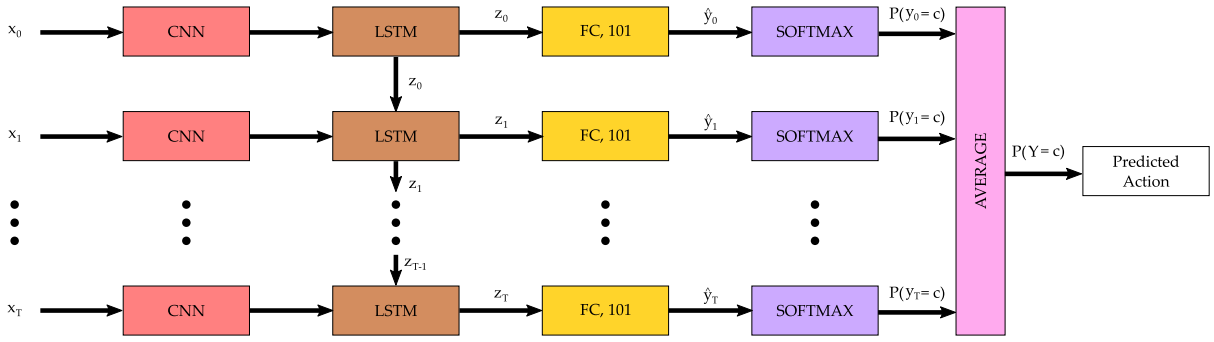


Figure 3: Sequence Learning and Action Prediction stage.

The output of this system is the average probability distribution between all per-frame probabilities. Therefore, the predicted action will correspond to the highest value of the average distribution.

The LRCN is end-to-end trainable, so its two stages are jointly trained using SGD with momentum and considering RGB and optical flow as inputs. For both training and testing, the UCF101 dataset [2] has been used.

The results of the experiments are reported in table 1 and suggest that the performance of the LRCN is better than that of the baseline for both RGB and flow inputs, concluding that the idea of learning the temporal dynamic of the visual features is beneficial. Particularly, the best results are obtained when fusing RGB and flow trained systems, achieving a 84.12% of classification accuracy.

	Single Network		**Weighted Average**	
	RGB	**Flow**	**1/2, 1/2**	**1/3, 2/3**
Baseline Model	67.37	74.37	75.46	78.94
LRCN Architecture	68.20	78.47	81.56	84.12

Table 1: Average accuracy of the baseline and the LRCN system across the three splits of the UCF101 test set, with RGB and flow as inputs.

Results in table 2 demonstrate that when compared to other state-of-the-art alternatives, the implemented system obtains a comparable accuracy value.

Method	**Accuracy**
Slow Fusion [3]	65.4%
Spatial-Stream [4]	73.0%
LRCN Architecture	84.12%
Two-Stream [4]	86.9%
CNN + LSTM [5]	88.6
IDT [6]	89,1%
C3D[7]	90.4%
LTC [8]	91.7%

Table 2: Average accuracy of state-of-the-art methods. LRCN architecture shows a performance comparable to the state of the art. The information within this table has been obtained from [8] and [4]

Acronyms

AI Artificial Intelligence.

ANN Artificial Neural Network.

CNN Convolutional Neural Network.

FNN Feedforward Neural Network.

GMM Gaussian Mixture Model.

GPU Graphics Processing Unit.

HDP Hierarchical Dirichlet Process.

HMM Hidden Markov Model.

IDT Improved Dense Trajectories.

ILSVRC ImageNet Large Scale Visual Recognition Challenge.

LDA Latent Dirichlet Allocation.

LRCN Long-term Recurrent Convolutional Network.

LSTM Long Short-Term Memory.

MDT Mixture of Dynamic Texrures.

MLP Multi-Layer Perceptron.

MPPCA Mixture of Probabilistic Principal Component Analysis.

MRF Markov Random Field.

NN Neural Network.

ReLU Rectified Linear Unit.

RNN Recurrent Neural Network.

SCD Structural Context Descriptor.

SGD Stochastic Gradient Descent.

SSMF Sparse Semi-non-negative Matrix Factorization.

SVM Support Vector Machine.

Contents

Agradecimientos v

Resumen vii

Abstract ix

Extended Abstract xi

Acronyms xv

1 Introduction 1

1.1 Motivation and Objective 1

1.2 Document Outline 3

2 State of the Art 5

2.1 General Aspects of the Anomaly Detection Problem 6

2.1.1 Types of Anomalies 6

2.1.2 Operation Modes 8

2.2 Related Work 10

2.2.1 Approaches Based on Trajectories 10

2.2.2 Approaches Based on Low-level Features and Motion Representation . . 11

2.2.3 Approaches Based on Deep Features 12

2.3 Conclusion . . . 13

3 Convolutional and Recurrent Neural Networks 15

3.1 Fundamentals of Neural Networks . . . 16

3.1.1 Feedforward Networks . . . 17

3.1.2 Output Units . . . 18

3.1.3 Hidden Units . . . 19

3.1.4 Back-Propagation . . . 22

3.1.5 Regularisation . . . 25

3.1.6 Optimisation. Learning Parameters . . . 26

3.2 Convolutional Neural Networks . . . 28

3.2.1 The Convolution Operation . . . 29

3.2.2 CNN Architecture . . . 29

3.2.2.1 Convolutional Layer . . . 30

3.2.2.2 Pooling Layer . . . 32

3.2.2.3 Fully-Connected Layer . . . 33

3.2.3 Well Established CNN Architectures . . . 33

3.3 Recurrent Neural Networks . . . 34

3.3.1 Bidirectional Recurrent Neural Networks . . . 36

3.3.2 The Problem of Long-Term Dependencies . . . 36

3.3.3 Long-Short-Term Memory . . . 38

3.4 Conclusion . . . 40

4 Implementation: LRCN 41

4.1 Visual Feature Extraction . . . 42

4.2 Sequence Learning and Action Prediction . . . 43

4.3 Conclusion 45

5 Results **47**

5.1 Experimental Set-up 47

5.1.1 UCF101 Dataset 47

5.1.2 Baseline Method 48

5.1.3 Training the Model 48

5.2 Evaluation 50

5.3 Conclusion 51

6 Conclusion and Future Work **53**

6.1 Conclusion 53

6.2 Future Work 54

List of Figures

1.1 Long-term Recurrent Convolutional Network (LRCN) architecture [1]. 2

2.1 A simple two dimensional example of different cases of anomalies[21]. 7

2.2 Contextual anomaly t_2 in a temperature time series [19]. 8

2.3 Different anomaly detection modes [21]. 8

3.1 Analogy between a biological neuron and a mathematical model [62]. 17

3.2 A 3-layer fully-connected neural network [62]. 18

3.3 Sigmoid activation function [62]. 20

3.4 Hyperbolic tangent activation function [62]. 20

3.5 ReLU activation function [62]. 21

3.6 Use of dropout to a standard NN [12]. 26

3.7 Dimensions in a CNN [62]. 30

3.8 Local connectivity in a convolutional layer [62]. 31

3.9 Pooling Layer function [62]. 33

3.10 Recurrent Neural Network without outputs [12]. 35

3.11 Recurrent Neural Network with outputs [12]. 36

3.12 Bidirectional Recurrent Neural Network [12]. 37

3.13 Long-Short-Term Memory cell structure [69]. 39

4.1 Long-term Recurrent Convolutional Network architecture [1]. 41

4.2 CNN used in the visual feature extraction stage. 42

4.3 Sequence Learning and Action Prediction stage. 43

5.1 Sample frames for 6 action classes of UCF101 [2]. 48

5.2 Baseline used to compare the LRCN with. 49

List of Tables

5.1 Average accuracy of the baseline and the LRCN system across the three splits of the UCF101 test set, with RGB and flow as inputs. 50

5.2 Average accuracy of state-of-the-art methods. LRCN architecture shows a performance comparable to the state of the art. The information within this table has been obtained from [8] and [4] 50

Chapter 1

Introduction

Video surveillance can be defined as the process of monitoring or observing the activities, behaviour or movements of an individual or a group to provide enhanced security. It is used by retail stores, government departments, hospitals, transportation and logistics companies, and law enforcement agencies, among others, to prevent criminal activities. Due to the relevance of some of these possible scenarios, it has been an area of significant interest in both academia and industry [9]. Proof of the latter is the increase in the number of cameras and the amount of space under surveillance as well as the raise of capital invested in this field of research.

Looking at the most relevant journals and conferences in this field (chapter 2), it can be noticed that in the last decade, the attention devoted to this matter has massively grown, yielding a fair number of research contributions applicable in industry in the near future. Furthermore, the evolution of the cameras coupled with the irruption of new fast processing technologies such as *NVIDIA* [10] Graphics Processing Units (GPU), have led to a turning point in video and image processing.

1.1 Motivation and Objective

Usually, the surveillance task is carried out by security staff. The common situation would be a set of cameras properly allocated to cover a specific area, then, the staff would check the video to find if something occurs without following the expected behaviour. In other words, the personnel try to detect an ***anomaly***.

Therefore, following the aforementioned idea of video surveillance task, in this context, an anomaly can be defined as an action or a behaviour that deviates from the common rule. Usually, anomalies occur in a very small proportion of the time in which the task takes place. It is for this reason that this job might be very monotonous and boring for the person who

performs it [11], hence making the video surveillance task an obvious candidate to become an automated task.

Regarding the human behaviour context, an anomaly would appear when an individual performs an action that does not correspond with the usual behaviour in the controlled area. Thus, a possible anomaly detection system could include a **human activity recognition system**.

Within this context, the objective of this Master's Thesis is, first, to review the existing work in anomaly detection and, then, to study and implement a state-of-the-art human activity recognition system. It is based on the Long-term Recurrent Convolutional Network (LRCN) architecture proposed by [1]. As figure 1.1 indicates, this design benefits from both the good performance of Convolutional Neural Networks (CNN) in visual features extraction and the strength of learning sequence information of Recurrent Neural Networks (RNN), specifically Long Short-Term Memory (LSTM) recurrent modules.

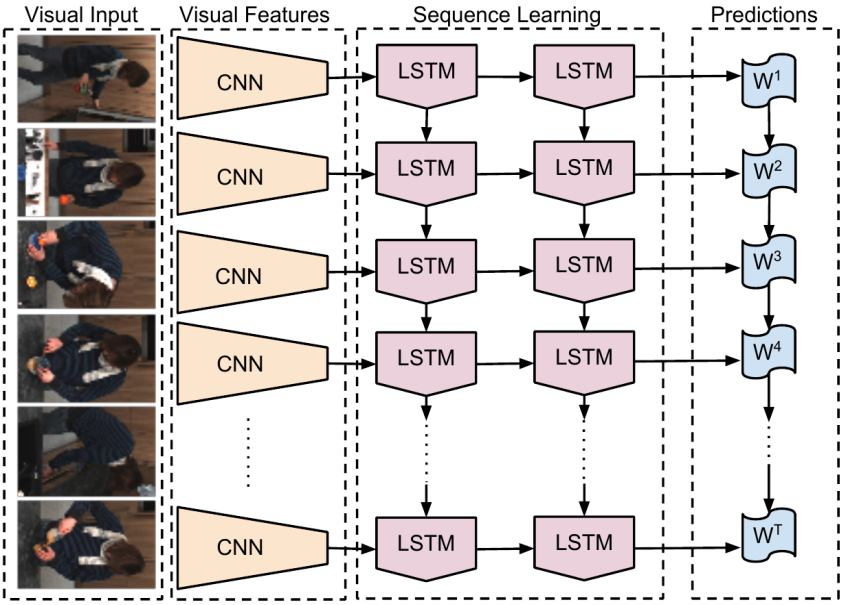


Figure 1.1: Long-term Recurrent Convolutional Network (LRCN) architecture [1].

As specified in [12], CNN are a specialized kind of Neural Network (NN) for processing data that has a known, grid-like topology, such as time-series data, which can be thought of as 1D grid, and image data, which can be thought of as a 2D grid of pixels. The name of *convolutional neural network* indicates that the network employs the convolution operation in place of general matrix multiplication.

On the other hand, LSTM networks are a kind of RNN capable of learning long-term dependencies. LSTM units have hidden state augmented with non-linear mechanisms to allow state to propagate without modification, be updated or be reset, using simple learned gating functions.

The system is made with the widely adopted deep learning framework *Caffe* [13]. This is a

fully open-source framework that affords clear access to deep architectures. Its code is written in C++ with CUDA [14] used for GPU computation.

All the details about the implementation of the proposed system are given in the following chapters.

1.2 Document Outline

This document is organised as follows:

- In **Chapter 2**, a review of the existing work in the field of anomaly detection is made. In the recent years, different lines of research have been explored to address the anomalous human behaviour detection problem. This section clarifies those lines to contextualise the work presented in this document.
- **Chapter 3** shows the fundamentals of NN and a detailed explanation of both CNN and RNN. Due to their benefits, convolutional networks have been widely used in many computer vision works. However, even though recurrent networks, and more specifically LSTM, are early ideas from the 80s and the 90s respectively [15], [16], only in the recent decade did researchers start to show a growing interest.
- The implementation of the Long-term Recurrent Convolutional Network architecture shown in figure 1.1 is explained elaborately in **Chapter 4**. This section describes how the sequence learning phase deals with the features extracted by the CNN to give a prediction of the action that it is being perform within a certain video.
- The performance of the LRCN is analysed in **Chapter 5**. The system is evaluated on an accepted action recognition benchmark, obtaining the mean classification accuracy. Then, the results are compared to a baseline model and to other state-of-the-art methods.
- After presenting all the work, **Chapter 6** shows some conclusions and possible future works.

Chapter 2

State of the Art

Anomaly detection in video surveillance scenes has been a challenging problem for Computer Vision and Machine Learning researchers, mainly due to the wide variety of interesting applications such as examination of crowd motion, traffic flow monitoring, biometric identification or human behaviour interpretation in retail spaces. In video surveillance context, and more specifically in human behaviour analysis, an anomaly can be define as an action or a behaviour that deviates from the expected pattern.

However, there are many other fields where similar detecting algorithms can be applied [17] like, for example, network security, credit card fraud or illegal transactions in banking (see also [18]–[20] for more information about application domains). Despite the fact that all mentioned scenarios are clearly different, a common definition of anomaly can be applied to all of them: anomaly detection is the process of identifying unexpected items or events in datasets which differ from the norm [21].

Consequently, the common ground of every approach is to translate the problem into a data representation space and, then, interpret it. The key is to find the algorithm which obtains the most meaningful data and performs the best interpretation thereof. Regarding video analysis, the data representation would be made by extracting features from video sequence, then the interpretation would be to decide which of those features correspond to a normal pattern or an anomaly.

In this chapter, a review of the existing work in the literature of anomaly detection is made. First of all, some basics about anomalies are presented so as to understand the different set-ups that can be implemented to detect anomalies in datasets. Afterwards, the main lines of research in the problem of detecting anomalous events in video surveillance scenes that the scientific community has proposed over the last decade are analysed.

2.1 General Aspects of the Anomaly Detection Problem

Each specific solution for an anomaly detection problem is determined by two important aspects: the type of anomaly and the availability of labels [19]. There are also other aspects such as constraints and requirements of the application that will influence the solution, although they are not as crucial as the first ones.

2.1.1 Types of Anomalies

Given any data set representation as the one shown in the figure 2.1, the main idea of an anomaly detection algorithm is to detect data instances which deviate from the norm. However, the meaning of anomaly depends on how the problem is analysed. Hence, different kinds of anomalies can be defined.

Global and Local Anomalies

Figure 2.1 illustrates some cases using a simple two-dimensional dataset. Two anomalies can be easily identified by eye: x_1 and x_2 are very different from the dense areas with respect to their attributes and are therefore called ***global anomalies***. When looking at the dataset globally, x_3 can be seen as a normal record since it is not too far away from cluster c_2. However, if one focuses only on cluster c_2 and compares it with x_3 while neglecting all the other instances, it can be seen as an anomaly. Therefore, x_3 is called a ***local anomaly***, since it is only anomalous when compared with its close-by neighbourhood. It depends on the application, whether local anomalies are of interest or not.

Point, Collective and Contextual Anomalies

Another interesting question is whether the instances of cluster c_3 should be seen as three anomalies or as a small regular cluster. This phenomena is called *micro cluster* and anomaly detection algorithms should assign scores to its members larger than the normal instances, but smaller values than the obvious anomalies. This simple example already illustrates that anomalies are not always obvious and a score is much more useful than a binary label assignment.

To this end, an anomaly is always referred to a single instance in a dataset only occurring rarely. In reality, this is often not true. For example, in intrusion detection, anomalies are often referred to many (suspicious) access patterns, which may be observed at a larger amount as normal accesses. The task of detecting single anomalous instances in a larger dataset, as introduced so far, is called ***point anomaly detection***.

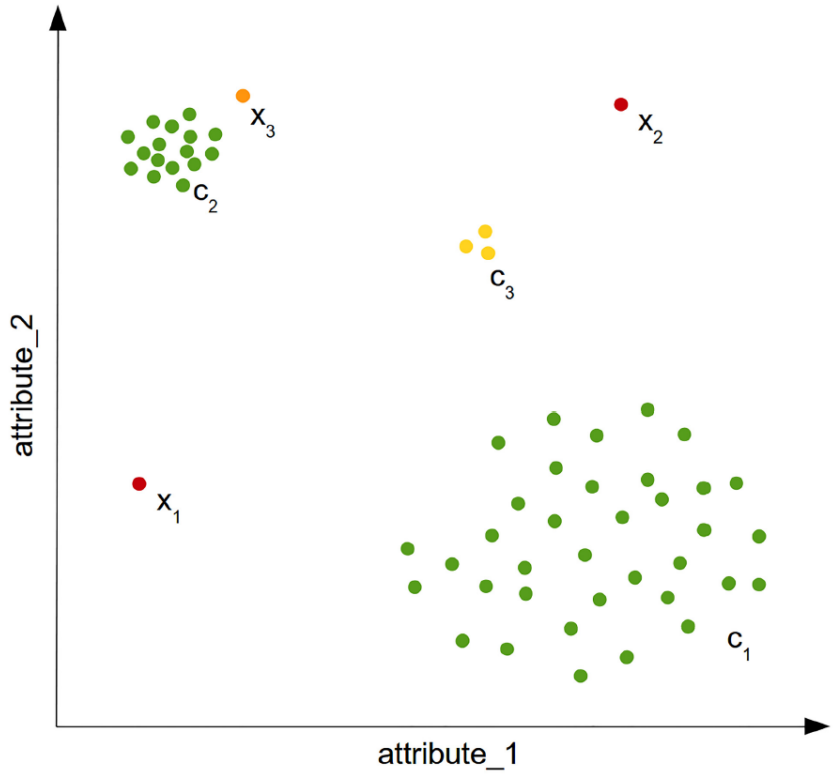


Figure 2.1: A simple two dimensional example of different cases of anomalies[21].

If an anomalous situation is represented as a set of many instances, this is called a ***collective anomaly***. Each of these instances is not necessarily a point anomaly, but only a specific combination thereof defines the anomaly. The previous given example of occurring multiple specific access patterns in intrusion detection is such a collective anomaly.

A third kind are ***contextual anomalies***, which describe the effect of a point which can be seen as normal, but when a given context is taken into account, the point turns out to be an anomaly. The most commonly occurring context is time. Figure 2.2 represents a temperature time series which shows the monthly temperature of an area over the last few years. The temperature value at time t_1 might be normal during the winter but the same value during the summer (at time t_2) would be an anomaly. This is a clear example of a contextual anomaly.

However, it is possible to use point anomaly detection algorithms to detect contextual and collective anomalies. The way to do so is by increasing data dimensionality. The context itself can be included as a new feature. Regarding to the example shown in figure 2.2, the inclusion of the month as another feature could allow the problem to be solved with a point anomaly detection approach.

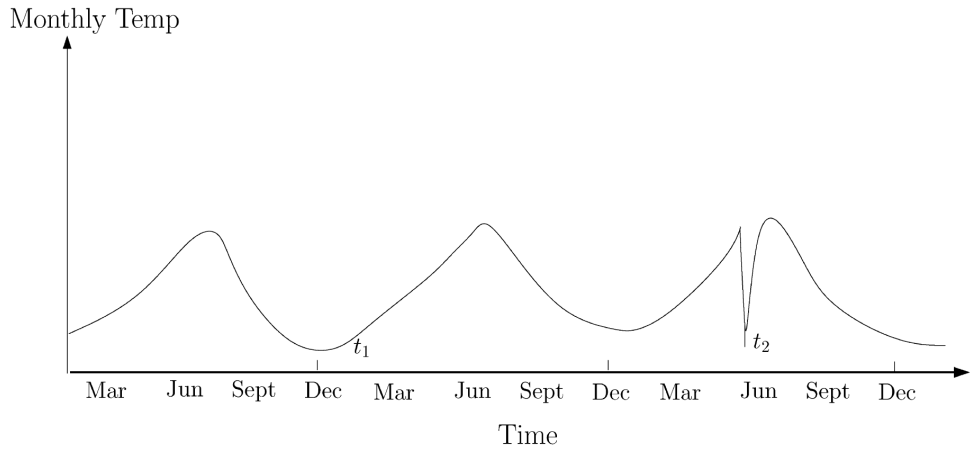
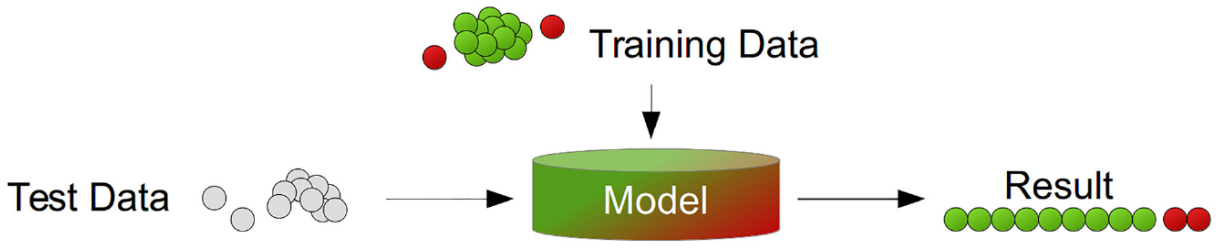


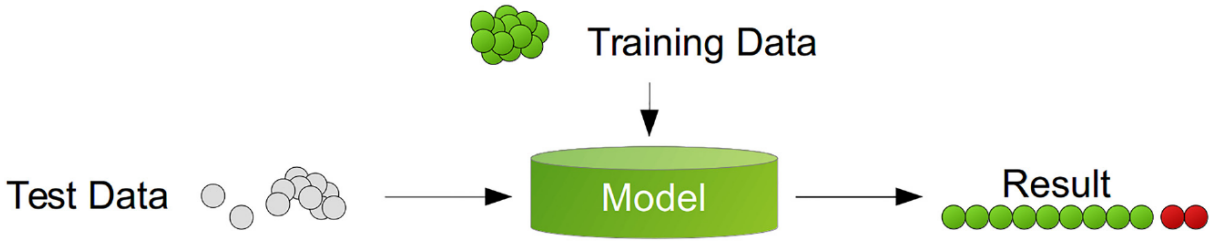
Figure 2.2: Contextual anomaly t_2 in a temperature time series [19].

2.1.2 Operation Modes

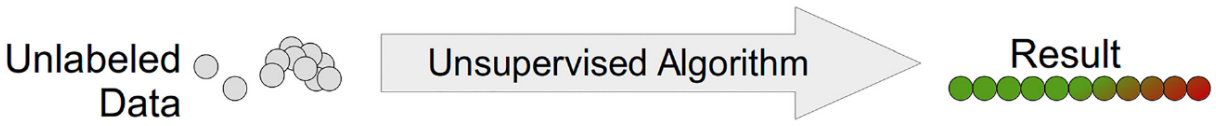
Basically, the anomaly detection set-up to be used depends on the labels available in the dataset. As figure 2.3 illustrates , three main types of set-ups can be distinguished [21].



(a) Supervised anomaly detection.



(b) Semi-supervised anomaly detection.



(c) Unsupervised anomaly detection.

Figure 2.3: Different anomaly detection modes [21].

Supervised Anomaly Detection

In this mode, the data comprises fully labelled training and test data sets. An ordinary classifier can be trained first and applied afterwards. This scenario is very similar to traditional pattern recognition with the exception that classes are typically strongly unbalanced.

In the context of this work, supervised methods build models of normal and abnormal behaviour based on labelled data. Video segments or frames that do not fit the models are considered as abnormal. Even though there are some past works using this set-up [22]–[25], it is barely used to this day since it assumes that anomalies are known and correctly labelled, and, in many applications, anomalies are not known in advance or may occur spontaneously during the task.

Semi-supervised Anomaly Detection

This set-up also uses training and test datasets, whereas training data only gathers normal data without anomalies. The basic idea is that a model of the normal class is learned and anomalies can be detected afterwards by deviating from that model.

When it comes to anomalous event detection, the semi-supervised set-up consists in a previous supervised training of a model [26]–[34], dictionary [35], [36] or classifier [37], [38] of normal events in the scene. Then, an event is anomalous if it does not fit the model, the features do not correspond with the dictionary or the classifier does not classify the event as normal.

As explained, in this set-up, a training phase previous to the detection task is required. Although this method has fewer limitations than completely supervised algorithms, this could only be used in scenarios where the normal events are very well known and are always the same. However, if the scene changes over time, new normal events could appear. Consequently, new training would be needed.

Unsupervised Anomaly Detection

As previously mentioned, if a semi-supervised set-up is applied in a changing environment, training is likely to be required over and over. This case leads to thinking of an algorithm capable of learning automatically without using any labelled data. To this end, the unsupervised set-up is the solution.

An unsupervised set-up is the most flexible set-up since it does not require any labels. The idea is that an unsupervised anomaly detection method scores the data solely based on the properties of the dataset. Typically, distances or densities (clustering) are used to give an es-

timate of what is normal and what is an outlier. Despite semi-supervised set-up being fairly used, unsupervised algorithms are being extremely studied and applied. Due to the random nature of anomalous events, this set-up is what best fits to the anomaly detection problem.

Regarding video surveillance, there are many works in the literature performing the anomaly detection task with an unsupervised method. The most common architecture is a two-stage system. The first stage extracts the features of the video, while the second applies an algorithm to determine whether an anomaly exists or not. These algorithms can be based on clustering [39]–[45], codebooks and bag of words [46]–[48] or probability models [9], [49]–[52], among others. With the recently growing interest in deep learning algorithms, a few works have also begun to use unsupervised set-ups with different types (NN) architectures.

2.2 Related Work

Within the last two decades, a lot of work has been developed where many different ways of addressing the problem presented in this document have been proposed. In spite of the wide variety of statistical and image processing techniques that have been used, one can distinguish between two main lines of research: trajectory-based and low-level features approaches. Nevertheless, as a result of the growing popularity of deep learning methods, the issue is currently being worked out with varying architectures and types of NNs, such as CNNs, RNNs or autoencoders. Consequently, a quite open new line of research based on deep features must be discerned.

2.2.1 Approaches Based on Trajectories

The main idea of such approaches is the analysis and the modelling of normal trajectories obtained by tracking individual objects in training videos, then declaring behaviours to be abnormal when they deviate from normal tracks.

All the previously mentioned set-ups can be applied. In both supervised and semi-supervised set-ups, the normal trajectories are learned from labelled training data. For instance, in [37] in the training stage, normal trajectories are obtained to train a Gaussian Mixture Model-based (GMM) classifier. During the surveillance task, all the trajectories with sufficiently low likelihood with respect to the model can be flagged as potentially anomalous. The same happens in [31], but using GMM to model regions of interest in the scene and a particle filter in tracking. Other supervised and semi-supervised methods with trajectory-based approaches are presented in [24], [33], respectively.

There are also works that proposed unsupervised algorithms. The main structure of these

methods consists of a first phase to obtain trajectories, and a second phase whereby anomalous trajectories are detected with clustering ([39], [40], [43], [44]) or probabilistic models ([49], [53]) algorithms. Support Vector Machines (SVM) or spectral clustering as well as Mixtures of Probabilistic Principal Component Analysis (MPPCA) are examples of clustering methods used in the literature. On the other hand, Hidden Markov Models (HMM) and GMM are probabilistic model algorithms commonly used.

Even though these trajectory-based methods can achieve good results when foreground objects are easy to detect and track, they suffer from occlusion problems, and more so in crowded scenes. That is why many other researchers have proposed to work with lower level algorithms.

2.2.2 Approaches Based on Low-level Features and Motion Representation

So as to avoid the limitation of anomaly detection algorithms based on trajectories, some methods have been introduced using low-level features, such as gradients or pixel motion. These methods attempt to learn and model shapes and spatio-temporal relations using low-level feature distributions.

Authors in [29], introduce a joint detector of temporal and spatial anomalies. This work uses a Mixture Dynamic of Textures (MDT) for modelling the normal crowd behaviour and events are considered as anomalous when they represent outliers with respect to the model. Authors use discriminant saliency to distinguish spatial anomalies from normal events. An extended version of this work has been provided in [54].

On the other hand, unusual events in [55] are detected by multiple monitors which use local and low-level features; whereas the work in [9] proposes a probabilistic method to handle local spatio-temporal anomalies. The authors use spatio-temporal features and a K-nearest neighbour method to detect anomalies among the video regions.

Behaviour is modelled in [27] by using pixel motion properties. For normal events, a spatio-temporal co-occurrence matrix is trained, then video data are represented by using this matrix and a Markov Random Field (MRF). This representation is then used to detect anomalies. Likewise, authors in [41] and [32] also use MRF, however, the former uses optical flow to find pixel motion patterns while the latter works with features of rarity, unexpectedness and irrelevance. Unlike previously mentioned works, [30] extracts pixel motion representation with spatio-temporal gradients and uses a HMM to detect anomalies.

There are some works that propose to use different types of descriptors containing multiple kinds of features such as, spatio-temporal energy, gradient histograms or agent interactions. A hierarchical Bayesian model which combines low-level visual features, simple “atomic” ac-

tivities and multi-agent interactions is developed by [22]. The proposed model includes improved Latent Dirichlet Allocation (LDA) and a Hierarchical Dirichlet Process (HDP) by using unsupervised modelling of interactions. The work presented in [56] introduces an informative Structural Context Descriptor (SCD) to represent a crowd; the spatio-temporal SCD variation of a crowd is analysed to localise anomaly regions. In this way, [28] proposes a method based on spatio-temporal directed energy filtering to model behaviours, detecting anomalies by using a histogram comparison method.

Different approaches have interpreted the problem of finding anomalies as a reconstruction and matching problem [25], [34]–[36], [38], [48], [50]. In a first step, the system learns the representation of the features, then, a new observation is considered an anomaly if its feature reconstruction does not match the learned representations.

In particular proposals in [25], [34], [38], [48] address the matching problem by learning set, dictionaries or codebooks with the representations of what is normal. For example, [34], [38] suggest to learn sets of normal representations and if a test set does not match the training sets, it is considered as an abnormal set, while [48] introduces a method to learn the events in video data by constructing a hierarchical code-book for dominant events in the video.

Conversely, sparse representations are used in [35], [36], [50]. A high speed rate method with good performance (140-150 FPS) is achieved by [36] using this kind of data representation. Furthermore, authors in [50] propose the Sparse Semi-non-negative Matrix Factorization (SSMF) for learning local patterns of pixels. Then a probability model using those local patterns of pixels is learned for considering both the spatial and temporal contexts. This method is totally unsupervised, and anomalies are detected by learned models.

2.2.3 Approaches Based on Deep Features

Recently, in the field of image processing, numerous works are being carried out using deep learning methods. In this video surveillance application, a few proposals make use of these powerful methods, showing that solutions based on deep learning are going to be extensively explored, forming therefore a new important line of research.

The tendency of approaches based on this kind of algorithms is to use architectures formed by CNN, RNN or autoencoders. These architectures are able to extract spatio-temporal and motion features, depending on how input data are delivered and how the system is trained.

The work presented in [52] divides each video in non-overlapping patches. The global descriptors of the video are learned by an autoencoder. [57] also divides the videos but uses the autoencoder to learn appearance representations of each frame.

In [58], a CNN receives the video patches to create local binary maps and thus obtain the motion representation. This information is also combined with the optical flow of video. Work reported in [59], [60] uses autoencoders to extract candidates of interest and then a CNN to evaluate them.

Another way to use deep learning methods is as in [61]. The proposed system learns spatial features with an autoencoder and temporal features with LSTM.

2.3 Conclusion

In this chapter, a deep analysis of the anomalous event detection problem has been made. At first, some basics aspects of anomalies have been introduced in order to understand the problem and the different applicable set-ups: supervised, semi-supervised and unsupervised. Currently, although there are some recently works with semi-supervised methods, unsupervised algorithms are the most popular.

Regarding the different approaches, trajectory-based and low-level features methods have traditionally been used. However, a new line of research in this field is beginning to be widely explored. Works in this new line are based on deep learning architectures such as CNN, RNN or autoencoders.

Consequently, this work proposes to use the LRCN, which is composed of a CNN and an LSTM. The idea is to learn the spatial and temporal features with the convolutional network and the LSTM, respectively.

Chapter 3

Convolutional and Recurrent Neural Networks

Today, ***Deep Learning*** is part of ***Artificial Intelligence*** (AI), which is a thriving field with many practical and active research topics. Many people have heard of deep learning as an exciting new technology. However, deep learning dates back to the 1940s. It only appears to be new, because it was relatively unpopular for several years preceding its current popularity, and because it has gone through many different names, and has only recently become called "deep learning".

Broadly speaking, there have been three waves of development of deep learning: deep learning known as ***cybernetics*** in the 1940s–1960s, deep learning known as ***connectionism*** in the 1980s-1990s, and the current resurgence under the name ***deep learning*** beginning in 2006 [12].

Some of the earliest learning algorithms we recognize today were intended to be computational models of biological learning, i.e. models of how learning happens or could happen in the brain. As a result, one of the names that deep learning has gone by is ***Artificial Neural Networks*** (ANN).

The neural perspective on deep learning is motivated by two main ideas. One idea is that the brain provides a proof by example that intelligent behaviour is possible, and a conceptually straightforward path to building intelligence is to reverse engineer the computational principles behind the brain and duplicate its functionality. Another perspective is that it would be deeply interesting to understand the brain and the principles that underlie human intelligence, so machine learning models that shed light on these basic scientific questions are useful, in addition to being fine engineering tools.

Today, neuroscience is regarded as an important source of inspiration for deep learning re-

searchers, but it is no longer the predominant guide for the field. Media accounts often emphasize the similarity of deep learning to the brain. While it is true that deep learning researchers are more likely to cite the brain as an influence than researchers working in other machine learning fields such as kernel machines or Bayesian statistics, one should not view deep learning as an attempt to simulate the brain. Modern deep learning draws inspiration from many fields, especially applied mathematics fundamentals like Linear Algebra, Probability, Information Theory, and Numerical Optimization.

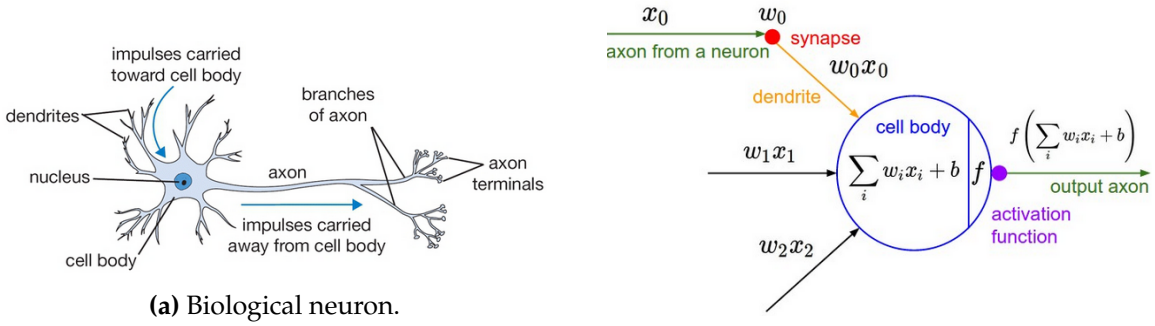
As figure 1.1 illustrates, the architecture of the solution is based on CNNs and LSTMs, which are different kinds of ANNs. In this chapter, a description of the theory on which the adopted solution is built in is given. It starts with an overview of the fundamentals of NNs, and it is followed by a deep explanation about convolutional and recurrent networks, specially long short-term memory units.

3.1 Fundamentals of Neural Networks

Following the aforementioned idea about the NN's origins, they have primarily been inspired by the goal of modelling biological neural systems, but have since diverged and become a matter of engineering and achieving good results in machine learning tasks.

The basic computational unit of the brain is a neuron. Approximately 86 billion neurons can be found in the human nervous system and they are connected with approximately 10^{14}–10^{15} synapses [62].

Figure 3.1 shows a biological neuron (3.1a) and a common mathematical model (3.1b). Each neuron receives input signals from its dendrites and produces output signals along its single axon. The axon eventually branches out and connects via synapses to dendrites of other neurons. In the computational model of a neuron, the signals that travel along the axons (e.g. x_0) interact multiplicatively, e.g. $w_0 x_0$, with the dendrites of the other neuron based on the synaptic strength at that synapse, e.g. w_0. The idea is that the synaptic strengths (the weights w) are learnable and control the strength of influence of one neuron on another and its direction: excitatory (positive weight) or inhibitory (negative weight). In the basic model, the dendrites carry the signal to the cell body where they all get summed. If the final sum is above a certain threshold, the neuron can fire, sending a spike along its axon. In the computational model, it is assumed that the precise timings of the spikes do not matter, and that only the frequency of the firing communicates information. Based on this rate code interpretation, the firing rate of the neuron can be modelled with an activation function f, which represents the frequency of the spikes along the axon.



(a) Biological neuron.

(b) Mathematical model.

Figure 3.1: Analogy between a biological neuron and a mathematical model [62].

3.1.1 Feedforward Networks

Deep feedforward networks, also often called Feedforward Neural Networks (FNN), or Multi-Layer Perceptrons (MLP), are the quintessential deep learning models. As described in [12], the goal of a feedforward network is to approximate some function f^* . For example, for a classifier, $y = f^*(\mathbf{x})$ maps an input $\mathbf{x}$ to a category y. A feedforward network defines a mapping $\mathbf{y} = f(\mathbf{x}; \theta)$ and learns the value of the parameters θ that result in the best function approximation.

These models are called feedforward because information flows through the function being evaluated from $\mathbf{x}$, through the intermediate computations used to define f, and finally to the output $\mathbf{y}$.

NNs are often organized into distinct layers of neurons. Mathematically, the model can be seen as group of functions connected in a chain, for example $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$. In this case $f^{(1)}$ is called the **first layer** of the network, $f^{(2)}$ is called the second layer, and so on.

The overall length of the chain gives the **depth** of the model. The final layer of a feedforward network is called the **output layer**. During neural network training, the algorithm drives $f(\mathbf{x})$ to match $f^*(\mathbf{x})$. The training data provides us with noisy, approximate examples of $f^*(\mathbf{x})$ evaluated at different training points. Each example $\mathbf{x}$ is accompanied by a label $y \approx f^*(\mathbf{x})$. The training examples specify directly what the output layer must do at each point x; it must produce a value that is close to y. The behaviour of the other layers is not directly specified by the training data. The learning algorithm must decide how to use those layers to produce the desired output, but the training data does not say what each individual layer should do. Instead, the learning algorithm must decide how to use these layers to best implement an approximation of $f^*(\mathbf{x})$. Because the training data does not show the desired output for each of these layers, these layers are called **hidden layers**.

Each hidden layer of the network is typically vector-valued. The dimensionality of these hidden layers determines the **width** of the model. Consequently, one should think of the layer

as consisting of many units that act in parallel, each representing a vector-to-scalar function.

From a less mathematical point of view [62], NNs can be seen as collections of neurons that are connected in an acyclic graph. In other words, the outputs of some neurons can become inputs to other neurons. For regular neural networks, the most common layer type is the fully-connected layer in which neurons between two adjacent layers are fully pairwise connected, but neurons within a single layer share no connections. An example of this network is presented in figure 3.2.

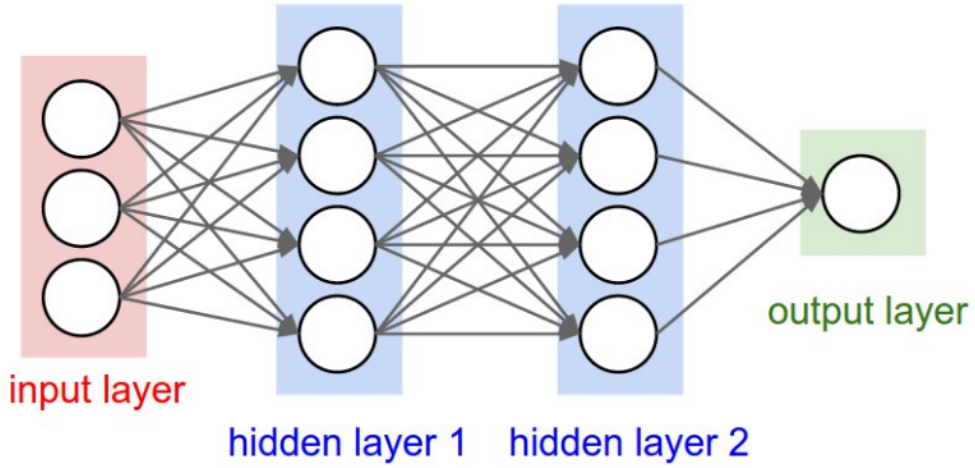


Figure 3.2: A 3-layer fully-connected neural network [62].

3.1.2 Output Units

Unlike all layers in an NN, the output layer neurons most commonly do not have an activation function. This is because the last output layer is usually taken to represent the class scores, e.g. in classification which are arbitrary real-valued numbers, or in regression which are some kind of real-valued target.

There are three major kinds of output units: **linear, sigmoid and softmax units**. The first one is the simplest and it is based on an affine transformation without non-linearity. Sigmoid units can be used in classification problems with two classes because they defined a Bernoulli distribution over y conditioned to x. The softmax unit is the one that is used in this work, therefore it is explained in a little more detail.

Softmax Output

Any time a probability distribution has to be represented over a discrete variable with n possible values, a softmax function may be used. These functions are most often used as the output of a classifier, to represent the probability distribution over n different classes.

With this purpose, a vector $\hat{\mathbf{y}}$ with $\hat{y}_i = P(\mathbf{y} = i|\mathbf{x})$ must be produced. It is required not only that each element of y_i be between 0 and 1, but also that the entire vector sums to 1 so that it represents a valid probability distribution. First, a linear layer predicts unnormalised log probabilities:

$$\mathbf{z} = \mathbf{W}^T\mathbf{h} + \mathbf{b} \tag{3.1}$$

where $z_i = log(\tilde{P}(y = i|\mathbf{x})$. The softmax function can then exponentiate and normalise $\mathbf{z}$ to obtain the desired $\hat{\mathbf{y}}$. Formally, the softmax function is given by:

$$softmax(\mathbf{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{3.2}$$

The use of the exp function works very well when training the softmax output to a target value y using maximum log-likelihood. In this case, the objective is to maximise the expression 3.3. Defining the softmax in terms of exp is natural because the log in the log-likelihood can undo the exp of the softmax, remaining as equation 3.4.

$$log(P(y = i; \mathbf{z})) = \log(softmax(\mathbf{z})_i) \tag{3.3}$$

$$\log(softmax(\mathbf{z})_i) = z_i - \log(\sum_j e^{z_j}) \tag{3.4}$$

3.1.3 Hidden Units

Generally, hidden units can be described as accepting a vector of inputs $\mathbf{x}$, computing an affine transformation as in 3.1 and then applying an element-wise non-linear function $g(\mathbf{z})$ called **activation function**.

The design of hidden units is an extremely active area of research and does not yet have many definitive guiding theoretical principles. It is usually impossible to predict in advance which will work best. The design process consists of trial and error, intuiting that a kind of hidden unit may work well, and then training a network with that kind of hidden unit and evaluating its performance on a validation set.

Every activation function, or non-linearity, takes a single number and performs a certain fixed mathematical operation on it. There are three major activation functions: sigmoid, hyperbolic tangent and rectified linear functions, among others.

Sigmoid and Hyperbolic Tangent Activation Functions

Historically, most NNs used the sigmoid and the tangent activation functions, which are defined by equations 3.5 and 3.6, and shown in figures 3.3 and 3.4, respectively. In addition, equation 3.7 demonstrates that these activation functions are closely related.

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{3.5}$$

$$g(z) = \tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{3.6}$$

$$\tanh(z) = 2\sigma(2z) - 1 \tag{3.7}$$

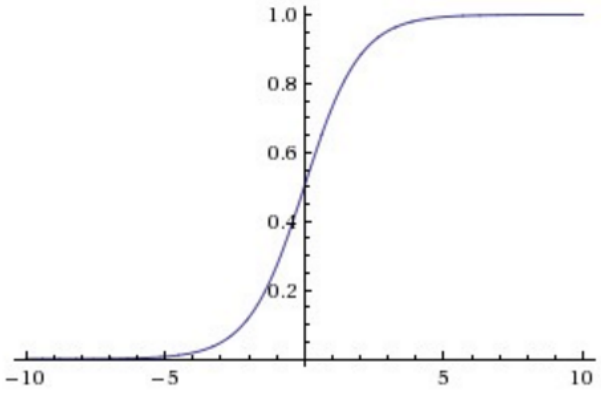


Figure 3.3: Sigmoid activation function [62].

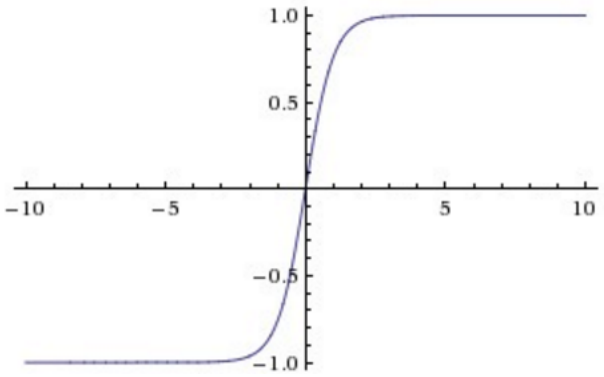


Figure 3.4: Hyperbolic tangent activation function [62].

Sigmoidal units saturate across most of their domain—they saturate to a high value when z is very positive, saturate to a low value when z is very negative, and are only strongly sensitive to their input when z is near 0. The widespread saturation of sigmoidal units can make gradient-based learning very difficult. For this reason, their use as hidden units in feedforward networks is now discouraged.

When a sigmoidal activation function must be used, the hyperbolic tangent activation function typically performs better than the logistic sigmoid. It resembles the identity function more closely, in the sense that $\tanh(0) = 0$ while $\sigma(0) = \frac{1}{2}$. Because $\tanh$ is similar to the identity function near 0, training a hyperbolic tangent unit is easier than a sigmoid one.

Rectified Linear Units

The Rectified Linear Unit (ReLU) has become very popular in the last few years. It computes the equation 3.8. In other words, the activation is simply thresholded at zero, as figure 3.5 shows.

$$g(z) = max\{0, z\} \tag{3.8}$$

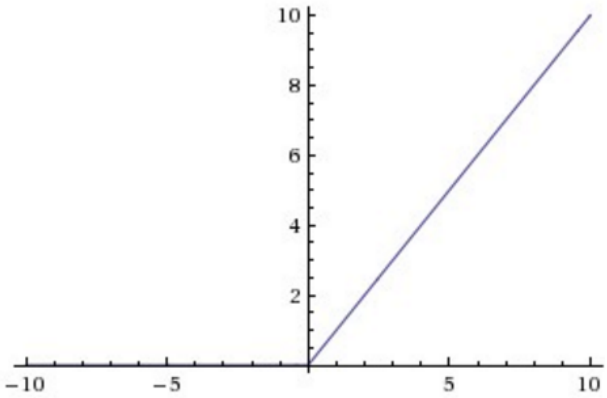


Figure 3.5: ReLU activation function [62].

A ReLU outputs zero across half its domain. This makes the derivatives through a rectified linear unit remain large whenever the unit is active. The gradients are not only large but also consistent. The second derivative of the rectifying operation is 0 almost everywhere, and the derivative of the rectifying operation is 1 everywhere that the unit is active. This means that the gradient direction is far more useful for learning than it would be in other activation functions.

One drawback to ReLU is that they cannot learn via gradient-based methods on examples in which their activation is zero. However, in practice, gradient descent still perform well enough for these units. This is in part because NN training algorithms do not usually arrive at a local minimum of the cost function, but instead merely reduce its value significantly. Nevertheless, there are some generalizations of ReLUs that try to resolve this limitation (**Leaky ReLU**, **Parametric ReLU** or **Maxout Units** [12]).

3.1.4 Back-Propagation

When a feedforward NN accepts an input $\mathbf{x}$ and produces an output $\hat{\mathbf{y}}$, information flows forward through the network. The inputs $\mathbf{x}$ provide the initial information that then propagates up to the hidden units at each layer and finally produces $\hat{\mathbf{y}}$. This is called forward propagation. During training, **forward propagation** can continue onward until it produces a scalar cost $J(\theta)$. The **back-propagation** algorithm allows the information from the cost to flow backwards through the network. Actually, back-propagation refers only to the method for computing the gradient, while another algorithm, such as Stochastic Gradient Descent (SGD), is used to perform learning using this gradient.

Gradient Interpretation and Chain Rule of Calculus

The gradient is used in a NN to optimise a cost function at the output so as to select the weights which provide the best performance of the network. To understand the effect of the gradient, it is also necessary to understand the interpretation of the derivative of a function.

Suppose we have a function 3.9, where both x and y are real numbers. The derivative of this function is denoted as $f'(x)$ or as $\frac{dy}{dx}$. The derivative $f'(x)$ gives the slope of $f(x)$ at the point x. In other words, it specifies how to scale a small change in the input in order to obtain the corresponding change in the output, as seen in equation 3.10.

$$y = f(x) \tag{3.9}$$

$$f(x + \epsilon) \approx f(x) + \epsilon f'(x) \tag{3.10}$$

The derivative is therefore useful for minimising a function because it helps to know how to change x in order to make a small improvement in y. $f(x)$ can be reduced or increased by moving x in small steps according to the sign of the derivative. This technique is called **gradient descent** and is the basic of every gradient-based optimisation method such as SGD.

Regarding the **chain rule of calculus**, it is used to compute the derivatives of functions formed by composing other functions whose derivatives are known. Back-propagation is an algorithm that computes the chain rule in a highly efficient way.

Let x be a real number, let f and g both be functions mapping from a real number to a real

number and suppose that $y = g(x)$ and $z = f(g(x)) = f(y)$. Then the chain rule states that:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \tag{3.11}$$

It can be generalised beyond the scalar case. Assuming that $\mathbf{x} \in \mathbb{R}^m$, $\mathbf{y} \in \mathbb{R}^n$, g maps from $\mathbb{R}^m$ to $\mathbb{R}^n$ and f maps from $\mathbb{R}^n$ to $\mathbb{R}$. If $\mathbf{y} = g(x)$ and $z = f(\mathbf{y})$, then:

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i} \tag{3.12}$$

In vector notation, this may be equivalently written as:

$$\nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)^{\top} \nabla_y z \tag{3.13}$$

where $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ is the $n \times m$ Jacobian matrix of g.

From this, it can be seen that the gradient of a variable $\mathbf{x}$ is obtained by multiplying a Jacobian matrix $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ by a gradient $\nabla_x z$. The back-propagation algorithm consists of performing such a Jacobian-gradient product for the whole chain of functions that represents the network.

Back-Propagation Algorithm in Neural Networks

To clarify the algorithm of back-propagation computation, one may consider the simplest case, as it could be a single-layer NN, and then generalise it to any depth.

Algorithm 3.1 first shows the forward propagation, which maps parameters to the supervised loss $L(\hat{\mathbf{y}}, \mathbf{y})$ associated with a single (input,target) training example $(\mathbf{x}, \mathbf{y})$, with $\hat{\mathbf{y}}$ the output of the NN when $\mathbf{x}$ is provided in input.

Algorithm 3.2 then shows the corresponding computation to be done for applying the back-propagation algorithm.

However, one can notice that these algorithms form a naive approach of the back-propagation method, since it has not been explained how to control the memory consumption of back-propagation. Usually, back-propagation involves the summation of high-dimensional components in long chains of functions to compute the gradient, thus resulting an overly high memory bottleneck. The way to avoid this will depend on the software in which the back-propagation is implemented.

Algorithm 3.1 Forward propagation through a typical deep neural network and the computation of the cost function. The loss $L(\hat{\mathbf{y}}, \mathbf{y})$ depends on the output $\hat{\mathbf{y}}$ and on the target $\mathbf{y}$. For simplicity, this demonstration uses only a single input example x. Practical applications should use a minibatch (as SGD) [12].

Require: Network depth, l
Require: $\mathbf{W}^{(i)}, i \in \{1, ..., l\}$
Require: $b^{(i)}, i \in \{1, ..., l\}$
Require: $\mathbf{x}$, the input to process
Require: $\mathbf{y}$, the target output
$\mathbf{h}^{(0)} = \mathbf{x}$
for $k = 1, ..., l$ **do**
$\mathbf{a}^{(k)} = \mathbf{b}^{(k)} + \mathbf{W}^{(i)}\mathbf{h}^{(k-1)}$
$\mathbf{h}^{(k)} = f(\mathbf{a}^{(k)})$
end for
$\hat{\mathbf{y}} = \mathbf{h}^{(l)}$
$J = L(\hat{\mathbf{y}}, \mathbf{y}) + \lambda\Omega(\theta)$

Algorithm 3.2 Backward computation for the deep neural network of Algorithm 3.1, which uses, in addition to the input x, a target y. This computation yields the gradients on the activations $\mathbf{a}^{(k)}$ for each layer k, starting from the output layer and going backwards to the first hidden layer. From these gradients, which can be interpreted as an indication of how each layer's output should change to reduce error, one can obtain the gradient on the parameters of each layer. The gradients on weights and biases can be immediately used as part of a SGD update (performing the update right after the gradients have been computed) or used with other gradient-based optimisation methods [12].

After the forward computation, compute the gradient on the output layer:
$\mathbf{g} \leftarrow \nabla_{\hat{\mathbf{y}}} J = \nabla_{\hat{y}} L(\hat{\mathbf{y}}, \mathbf{y})$
for k=l,l-1,...,1 **do**
Convert the gradient on the layer's output into a gradient into the pre-non-linearity activation (element-wise multiplication if f is element-wise):
$\mathbf{g} \leftarrow \nabla_{\mathbf{a}^{(k)}} J = \mathbf{g} \odot f'(\mathbf{a}^{(k)})$
Compute gradients on weights and biases:
$\nabla_{\mathbf{b}^{(k)}} J = \mathbf{g}$
$\nabla_{\mathbf{W}^{(k)}} J = \mathbf{g}\mathbf{h}^{(k-1)\top}$
Propagate the gradients to the next lower-level hidden layer's activations:
$\mathbf{g} \leftarrow \nabla_{\mathbf{h}} J = \mathbf{W}^{(k)\top}\mathbf{g}$
end for

Furthermore, real-world implementations of back-propagation also need to handle various data types, such as 32-bit floating point, 64-bit floating point or integer values. The policy for handling each of these types requires special care to design.

Another problem is that some operations have undefined gradients, and it is important to track these cases and determine whether the gradient requested by the user is undefined.

3.1.5 Regularisation

A central problem in machine learning is how to make an algorithm that will perform well not just on the training data, but also on new inputs (overfitting). Many strategies used in machine learning are explicitly designed to reduce the test error, possibly at the expense of increased training error. These strategies are known collectively as regularisation. There are several ways of control the capacity of a NN to generalise. Some of the most commonly used methods are described below.

L2 Regularisation

L2 regularisation is the most common form of regularisation. It can be implemented by penalising the squared magnitude of all parameters directly in the objective. That is to say, for every weight w in the network, a term $\frac{1}{2}\lambda w^2$ is added to the cost function, where λ is the regularisation strength. It is common to see the factor of $\frac{1}{2}$ in front because the gradient of this term with respect to the parameter w is, then, simply λw instead of $2\lambda w$. L2 regularisation has the intuitive interpretation of heavily penalising peaky weight vectors and preferring diffuse weight vectors. Lastly, one can notice that during gradient descent parameter update, using L2 regularisation ultimately means that every weight is decayed linearly.

L1 Regularisation

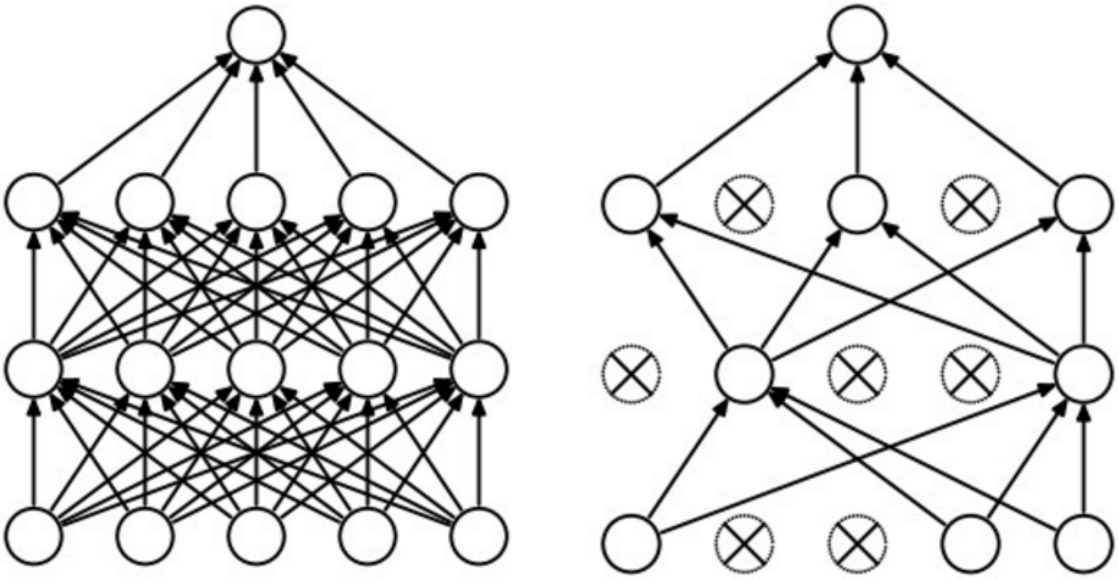
L1 regularisation is another relatively common form of regularisation, where for each weight w a term $\lambda|w|$ is added to the cost function. It is possible to combine L1 regularisation with L2 regularisation: $\lambda_1|w| + \lambda_2 w^2$ (this is called Elastic net regularisation). L1 regularisation has the intriguing property that it leads the weight vectors to become sparse during optimisation (i.e. very close to exactly zero). In other words, neurons with L1 regularization end up using only a sparse subset of their most important inputs and become nearly invariant to the "noisy" inputs. In comparison, final weight vectors from L2 regularisation are usually diffuse, small numbers.

Max Norm Constraints

Another form of regularisation is to enforce an absolute upper bound on the magnitude of the weight vector for every neuron and use projected gradient descent to enforce the constraint. In practice, this corresponds to performing the parameter update as normal, and then enforcing the constraint by clamping the weight vector $\vec{\mathbf{w}}$ of every neuron to satisfy $\|\vec{\mathbf{w}}\|_2 < \mathbf{c}$. One of its appealing properties is that the network cannot "explode" even when the learning rates are set too high because the updates are always bounded.

Dropout

Dropout is an extremely effective, simple and recently introduced regularisation technique that complements the other methods (L1, L2, Max Norm). While training, dropout is implemented by only keeping a neuron active with some probability p (a hyperparameter), or setting it to zero otherwise. Dropout can be interpreted as sampling an NN and only updating parameters of the sampled network based on the input data. An example of this method is illustrated in figure 3.6, where it can be seen that after applying dropout only a few neurons have remained active.



(a) Standard NN without dropout applied. **(b)** NN after applying dropout.

Figure 3.6: Use of dropout to a standard NN [12].

3.1.6 Optimisation. Learning Parameters

Up to now, with back-propagation one can obtain the gradient of the network in order to compute a certain cost function. The goal of training is to find the parameters of the network

(weights and biases) which perform the best optimisation of the cost function. To this aim, the training process can compute different optimisation algorithms.

Over time, several approaches for finding parameters have been proposed. The most common methods are gradient-based algorithms, specifically the SGD. This optimisation method and its variants are explained next.

Stochastic Gradient Descent

The SGD is an extension of the gradient descent algorithm (see section 3.1.4). A recurring problem in machine learning is that large training sets are necessary for good generalisation, but also computationally expensive.

The cost function used by a machine learning algorithm often decomposes as a sum over training examples of some per-example lost function. For example, the negative conditional log-likelihood of the training data could be:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} L(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \theta) \tag{3.14}$$

where L is the per-example loss $L(\mathbf{x}, \mathbf{y}, \theta) = -\log p(y|\mathbf{x}; \theta)$. For this cost function, the gradient would be:

$$\nabla_\theta J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta L(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \theta) \tag{3.15}$$

As the training set size grows, the time to take a single gradient step becomes prohibitively long. To avoid this problem, in SGD, the gradient is an expectation, and it is estimated using a small set of samples (**minibatch**) in each iteration. Now, the gradient is:

$$\mathbf{g} = \frac{1}{m'}\nabla_\theta \sum_{i=1}^{m'} L(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \theta) \tag{3.16}$$

And for each step, the parameters are estimated with:

$$\theta = \theta - \epsilon\mathbf{g} \tag{3.17}$$

where ϵ is the learning rate. In the equations, the learning rate is shown as a fixed value. However, it is important to notice that this value changes over the steps, in particular, it has to

decrease. Therefore, it allows to control how much the gradient affects to the parameters in the later steps of the algorithm.

Stochastic Gradient Descent with Momentum

The method of Momentum is designed to accelerate the learning. It is used as an additional hyperparameter of the SGD as it can be seen in equations 3.18 and 3.19.

$$\mathbf{v} = \alpha\mathbf{v} - \epsilon\frac{1}{m'}\nabla_\theta \sum_{i=1}^{m'} L(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \theta) \tag{3.18}$$

$$\theta = \theta + \mathbf{v} \tag{3.19}$$

The momentum algorithm accumulates an exponentially decaying moving average of the past gradients and continues to move in their direction. The hyperparameter α determines how quickly the contributions of previous gradients exponentially decay. The update velocity is represented by $\mathbf{v}$.

Stochastic Gradient Descent with Nesterov Momentum

Nesterov Momentum is a variant of the Momentum algorithm and it is also designed for accelerating the learning. The update equation is given by:

$$\mathbf{v} = \alpha\mathbf{v} - \epsilon\frac{1}{m'}\nabla_\theta \sum_{i=1}^{m'} L(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \theta + \alpha\mathbf{v}) \tag{3.20}$$

The difference between standard Momentum algorithm and Nesterov Momentum is where the gradient is evaluated. In Nesterov Momentum, the gradient is evaluated after the update velocity is applied.

3.2 Convolutional Neural Networks

Convolutional networks, also known as Convolutional Neural Networks (CNN) or ConvNets, are a specialised kind of NN for processing data that has a known, grid-like topology. Examples include time-series data, which can be thought of as a 1D grid taking samples at regular time intervals, and image data, which can be thought of as a 2D grid of pixels. Convolutional

networks have been tremendously successful in practical applications. The name "convolutional neural network" indicates that the network employs a mathematical linear operation called **convolution**. CNNs are simply NNs that use convolution in place of general matrix multiplication in at least one of their layers.

As previously stated, CNNs could work out in with promising results in any task involving grid-like data. In the present work, the inputs of these networks are images. Consequently, this section focuses on the characteristics of CNNs prepared to process images.

3.2.1 The Convolution Operation

In its most general form, convolution is an operation on two functions of real-valued argument. Expressions 3.21 and 3.22 define the continuous-time convolution. If the values of x and w are expressed at certain time intervals, the discrete-time convolution is defined by equation 3.23

$$s(t) = x(t) * w(t) \tag{3.21}$$

$$s(t) = \int x(\tau)w(t-\tau)d\tau \tag{3.22}$$

$$s[n] = \sum_{a=-\infty}^{\infty} x[k]h[n-k] \tag{3.23}$$

In convolutional network terminology, x is often referred to as the **input** and w as the **kernel**. The output can sometimes be called **feature map**. The input is usually a multidimensional array of data and the kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm.

3.2.2 CNN Architecture

CNNs take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular NN, the layers of a ConvNet have neurons arranged in 3 dimensions: **width**, **height**, **depth**. Moreover, the neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. Figure 3.7 shows a very simple diagram of a ConvNet.

A ConvNet, in the simplest case, is a sequence of layers that transform the image volume into an output volume. Three main types of layers are used to build ConvNet architectures:

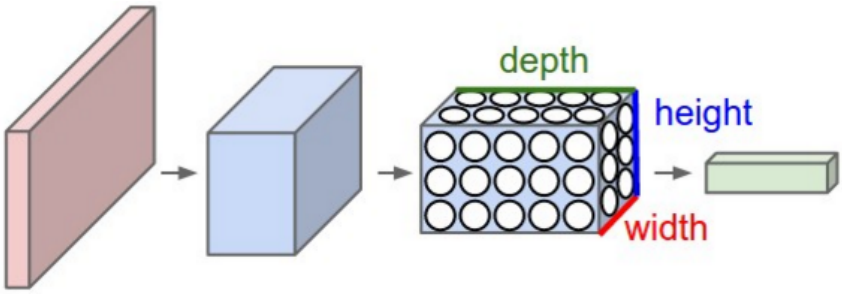


Figure 3.7: Dimensions in a CNN [62].

Convolutional Layer, **Pooling Layer**, and **Fully-Connected Layer**, among others. These layers are stacked to form a full ConvNet architecture.

3.2.2.1 Convolutional Layer

The convolutional layer is the core building block of a CNN which does most of the computational heavy lifting.

The convolutional layer's parameters consist of a set of learnable filters. Every filter is small spatially, along width and height, but extends through the full depth of the input volume.

During the forward pass, each filter is slid across the width and height of the input volume, computing dot products between the entries of the filter and the input at any position (convolution operation). As the filter is slid over the width and height of the input volume it produces 2D activation map that gives the responses of that filter at every spatial position.

Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some colour on the first layer, or eventually entire specific object patterns on higher layers of the network. This activation maps are stacked along the depth dimension to form the output volume.

The configuration and behaviour of the convolutional layer rely on three important aspects: **local connectivity**, **spatial arrangement** and **parameter sharing**. This three aspects are discussed below.

Local Conectivity

When dealing with high-dimensional inputs such as images, it is impractical to connect neurons to all neurons in the previous volume. Instead, each neuron is connected only to a local region of the input volume. The spatial extent of this connectivity is a hyperparameter called the **receptive field** of the neuron and it corresponds to the filter size. The extent of the connec-

tivity along the depth axis is always equal to the depth of the input volume. It is important to emphasise again this asymmetry in how the spatial dimensions are treated (width and height) and the depth dimension: the connections are **local in space** (along width and height), but always full along the entire depth of the input volume. An example of this property can be seen in figure 3.8, where each neuron in the convolutional layer is only connected to a local region in the input volume.

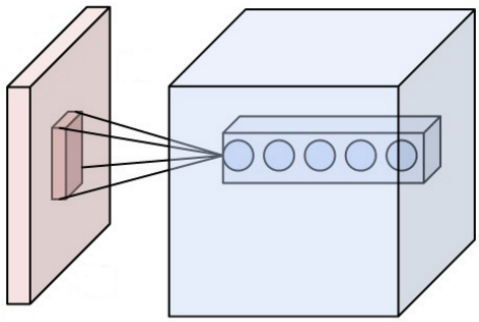


Figure 3.8: Local connectivity in a convolutional layer [62].

Spatial Arrangement

The previous property explains the connectivity of each neuron to the input volume. However, it does not specify how many neurons there are in the convolutional layer and how they are arranged. Three hyperparameters control the size of the output volume: **depth**, **stride** and **zero-padding**.

Depth. The depth of the output volume corresponds to the number of filters we would like to use, each of them learning to look for something different in the input. For example, if the first convolutional layer takes as input the raw image, then different neurons along the depth dimension may activate in presence of various oriented edged, or blobs of colour. The set of neurons that are looking at the same region of the input is known as a **depth column**.

Stride. The stride with which the filter is slid must be specified. When the stride is 1, the filters are moved one pixel at a time. When the stride is 2 (or uncommonly 3 or more, though this is rare in practice) then the filters jump 2 pixels at a time as they are slid. This will produce smaller output volumes spatially.

Zero-padding. Sometimes it will be convenient to pad the input volume with zeros around the border. The size of this zero-padding is a hyperparameter. The advantage of this hyperparameter is that it will allow for controlling the spatial size of the output volumes.

The spatial size of the output volume can be computed as a function of the input volume size (W), the receptive field size of the convolutional layer neurons (F), the stride with which they are applied (S), and the amount of zero-padding used (P) on the border. This function is

defined by expression 3.24.

$$\frac{W - F + 2P}{S} + 1 \tag{3.24}$$

Parameter Sharing

Parameter sharing scheme is used in convolutional layers to control the number of parameters. This amount can be reduced based on whether a feature is useful at some position in the image, then it should also be useful at a different position. Denoting a single 2D slice of depth as a **depth slice**, the neurons in each depth slice are going to be constrained to use the same weights and bias.

If all neurons in a single depth slice are using the same weight vector, then the forward pass of the convolutional layer can in each depth slice be computed as a convolution of the neuron's weights with the input volume.

From the memory footprint point of view, this parameter sharing will help to reduce the amount of memory used by the CNN.

3.2.2.2 Pooling Layer

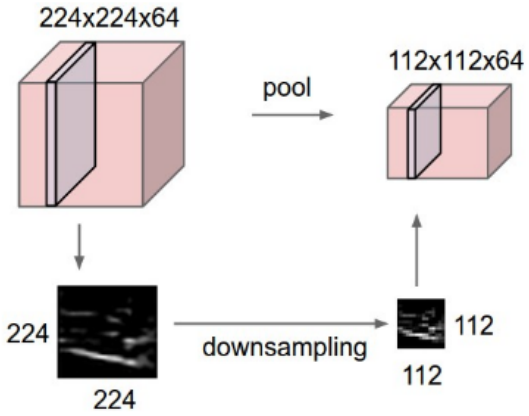
It is common to periodically insert a pooling layer in between successive convolutional layers in a CNN architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.

A pooling function replaces the output of a net at a certain location with a summary statistic of the nearby outputs. The most common pooling operation is **max pooling** but other popular functions are the **average** of a rectangular neighbourhood, the **L2-norm** of a rectangular neighbourhood or a **weighted average** based on the distance from the central pixel.

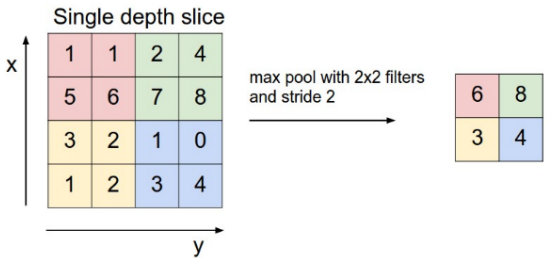
The pooling layer operates independently on every depth slice of the input and resizes it spatially. The most common form is a pooling layer with filters of size 2×2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations. Every MAX operation would in this case be taking a max over 4 numbers (little 2×2 region in some depth slice). The depth dimension remains unchanged.

In figure 3.9, there is an example of how the max pooling operation affects the input volume and a depth slice. 3.9a shows how the pooling downsamples the volume spatially, independently in each depth slice of the input volume and preserving the volume depth. On the other

hand, 3.9b illustrates the max pooling operation with a stride of 2 to a single depth slice. Each max is taken over 4 numbers.



(a) Effect of pooling in an output volume.



(b) Example of max pooling.

Figure 3.9: Pooling Layer function [62].

3.2.2.3 Fully-Connected Layer

The Fully-Connected layer is a traditional NN that uses a softmax output (other classifiers like SVM can also be used). In a ConvNet architecture, every neuron of the input volume is connected to every neuron of the next layer.

The output from the convolutional and pooling layers represents high-level features of the input image. The purpose of the fully-connected layer is to use these features for classifying the input image into various classes based on the training dataset.

Apart from classification, adding a fully-connected layer is also a cheap way of learning non-linear combinations of these features. Most of the features from convolutional and pooling layers may be good for the classification task, but combinations of those features might be even better.

In a classification task, the sum of output probabilities from the fully-connected layer should be 1. This is ensured by using the softmax output.

3.2.3 Well Established CNN Architectures

In the field of ConvNets, one rarely will have to train a new model from scratch. Instead of that, it is better to choose one network of the several architectures that have been thoroughly tested and finetune it to the specific task. The most common architectures are briefly presented next.

LeNet [63]. The first ConvNet architecture with succesfull applications was the LeNet. It was used to read zip codes, digits, etc.

AlexNet [64]. This ConvNet was a deeper and much wider version of the LeNet. It won by a large margin the difficult ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. It was a significant breakthrough with respect to the previous approaches and the current widespread application of CNNs can be attributed to this work.

ZFNet [65]. This network won the ILSVRC 2013. It became known as the ZFNet. It was an improvement on AlexNet by tweaking the architecture hyperparameters, in particular by expanding the size of the middle convolutional layers and making the stride and filter size on the first layer smaller.

GoogleNet [66]. The ILSVRC 2014 winner was a ConvNet from Google. Its main contribution was the development of an Inception Module that dramatically reduced the number of parameters in the network (4M, compared to AlexNet with 60M). Additionally, this paper uses average pooling instead of fully-connected layers at the top of the ConvNet, eliminating a large amount of parameters that do not seem to matter much.

VGGNet [67]. The runner-up in ILSVRC 2014 was the VGGNet. Its main contribution was showing that the depth of the network is a critical component for good performance. Their final best network contains 16 CONV/FC layers and, appealingly, features an extremely homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling from beginning to end. A downside of the VGGNet is that it is more expensive to evaluate and uses a lot more memory and parameters (140M).

ResNet [68]. The Residual Network was the winner of ILSVRC 2015. It features special skip connections and a heavy use of batch normalization. The architecture is also missing fully-connected layers at the end of the network. ResNets are currently by far state of the art ConvNets models.

3.3 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are a family of NNs for processing sequential data. Each neuron or unit can use its internal memory to maintain information about previous inputs.

These networks rely on the concept of parameter sharing, in the sense that a part of the model could be shared by different members of a sequence of data. This sharing is quite important when a particular piece of information in a specific order can occur in multiple positions within the sequence.

As considered before in section 3.1.1, in a feedforward neural network, the connections do not form cycles. However, if this condition is removed, allowing cyclical connections between the layers of the network, the concept of Recurrent Neural Networks comes up.

The graph in figure 3.10 represents a recurrent structure. It corresponds to a RNN with no outputs. The network processes information from the input x by incorporating it into the state h that is passed forward through time. Each state of the network is modelled by equation 3.25.

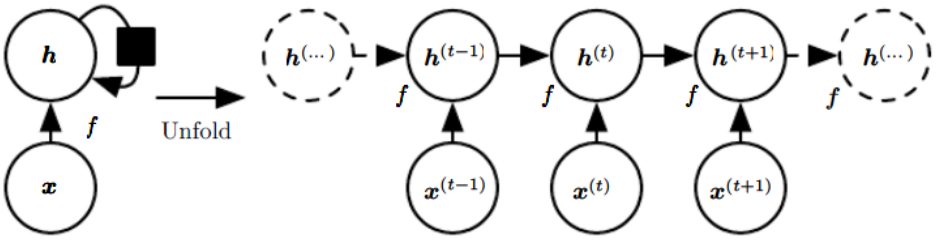


Figure 3.10: Recurrent Neural Network without outputs [12].

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}, \theta) \tag{3.25}$$

When the recurrent network is trained to perform a task that requires predicting the future from the past, the network typically learns to use $\mathbf{h}^{(t)}$ as a kind of lossy summary of the relevant task aspects of the past sequence of inputs up to t. Depending on the training criterion, this summary might selectively keep some aspects of the past sequence with more precision than other ones.

The above graph only shows how the information travels from the previous state to the next state, yet not mentioning anything about the output of the network. An example of an RNN considering outputs is illustrated in figure 3.11. The network has input to hidden connections parametrised by a weight matrix $\mathbf{U}$, hidden to hidden connections parametrised with a weight matrix $\mathbf{W}$ and hidden to output connections parametrised by a weight matrix $\mathbf{V}$. The estimates $\hat{\mathbf{y}}$ correspond to the softmax of the output of each state: $\hat{\mathbf{y}} = softmax(\mathbf{o})$.

Assuming, for example, that the activation function of the hidden layers is the hyperbolic tangent and knowing that the output is given by a softmax probability function, the forward propagation of the network could be modelled by the next set of equations.

$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} \tag{3.26}$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)}) \tag{3.27}$$

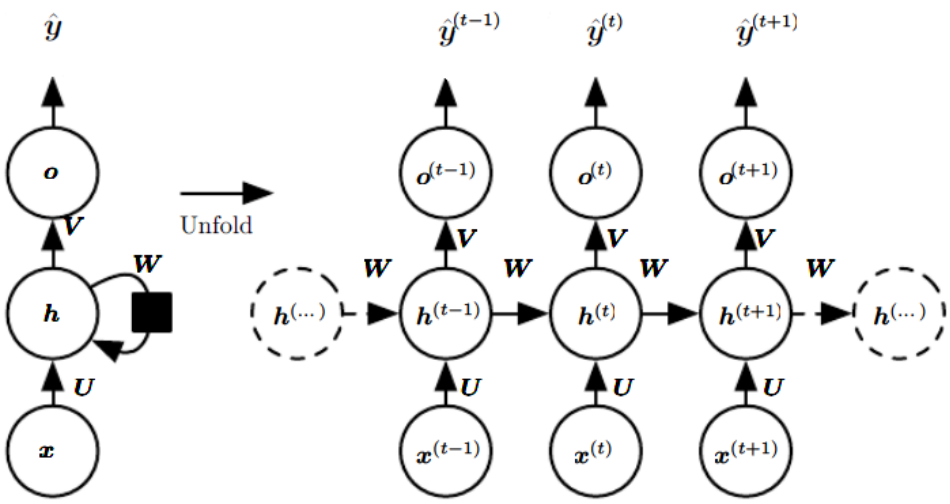


Figure 3.11: Recurrent Neural Network with outputs [12].

$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{U}\mathbf{V}^{(t)} \tag{3.28}$$

$$\hat{\mathbf{y}} = softmax(\mathbf{o}^{(t)}) \tag{3.29}$$

3.3.1 Bidirectional Recurrent Neural Networks

The RNN structure shown up to now, only considered that one state at a time t captures information from the past and the present. However, there are applications where the prediction depends on the whole input sequence (past, present and future). Bidirectional RNN were invented to address that need.

This kind of network combines an RNN that moves forward through time, beginning from the start of the sequence, with another RNN that moves backward through time, beginning from the end of the sequence. Figure 3.12 illustrates the typical bidirectional RNN, with $\mathbf{h}^{(t)}$ standing for the state of the sub-RNN that moves forward through time and $\mathbf{g}^{(t)}$ standing for the state of the sub-RNN that moves backward through time. This allows the output units $\mathbf{o}^{(t)}$ to compute a representation that depends on both **the past and the future**.

3.3.2 The Problem of Long-Term Dependencies

Recurrent networks involve the composition of the same function multiple times, once per time step. These compositions can result in extremely non-linear behaviour. In particular, the function composition employed by RNNs resembles a matrix multiplication, for example te

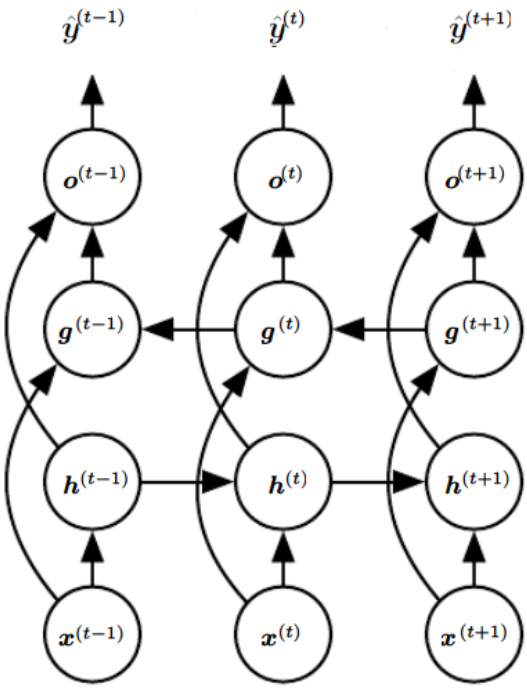


Figure 3.12: Bidirectional Recurrent Neural Network [12].

following relation:

$$\mathbf{h}^{(t)} = \mathbf{W}^{\top}\mathbf{h}^{(t-1)} \tag{3.30}$$

The latter expression is a very simple recurrent network lacking both a non-linear activation function and inputs $\mathbf{x}$. The recurrence relation essentially describes the power method shown in the next equation:

$$\mathbf{h}^{(t)} = (\mathbf{W}^{t})^{\top}\mathbf{h}^{(t-1)} \tag{3.31}$$

If $\mathbf{W}$ admits an eigendecomposition of the form of 3.32, with orthogonal $\mathbf{Q}$, the recurrence may be simplified further to expression 3.33.

$$\mathbf{W} = \mathbf{Q}\Lambda\mathbf{Q}^{\top} \tag{3.32}$$

$$\mathbf{h}^{(t)} = \mathbf{Q}^{\top}\Lambda^{t}\mathbf{Q}\mathbf{h}^{(0)} \tag{3.33}$$

The eigenvalues are raised to the power of t causing eigenvalues with magnitude less than one to **vanish** and eigenvalues with magnitude greater than one to **explode**. Any component

of $\mathbf{h}^{(0)}$ that is not aligned with the largest eigenvector will eventually be discarded. In the same way, gradients will be conditioned to this effect. As a result, there also exists the **vanishing an exploding gradient** problem.

As a consequence of this problem, simple RNNs are not able to learn long-term dependencies. Nevertheless, there are different methods to resolve this problem, such as leaky units or gated networks.

3.3.3 Long-Short-Term Memory

The idea of introducing self-loops to produce paths where gradient can flow for long durations (without vanishing or exploding) is a core contribution of the initial **Long-Short-Term memory** (LSTM)

Instead of a unit that simply applies an element-wise non-linearity to the affine transformation of inputs and recurrent units, LSTM recurrent networks have **LSTM cells** that have an internal recurrence (a self-loop), in addition to the outer recurrence of the RNN. Each cell has the same inputs and outputs as an ordinary recurrent network, but it has more parameters and a system of gating units that controls the flow of information.

The structure of an LSTM cell has the following stages and it is shown in figure 3.13.

1. **New Memory Generation**. Essentially, it uses the input $\mathbf{x}^{(t)}$ and the past hidden unit $\mathbf{h}^{(t-1)}$ to generate a new memory which includes aspects of the input.

$$\hat{c}^{(t)} = \tanh(\mathbf{W}^{(c)}\mathbf{x}^{(t)} + \mathbf{U}^{(c)}\mathbf{h}^{(t-1)}) \tag{3.34}$$

2. **Input Gate**. The input gate uses the input $\mathbf{x}^{(t)}$ and the past hidden state $\mathbf{h}^{(t-1)}$ to determine whether or not the input is worth preserving and thus is used to gate the new memory generation.

$$i^{(t)} = \sigma(\mathbf{W}^{(i)}\mathbf{x}^{(t)} + \mathbf{U}^{(i)}\mathbf{h}^{(t-1)}) \tag{3.35}$$

3. **Forget Gate**. The forget gate uses the input $\mathbf{x}^{(t)}$ and the past hidden state $\mathbf{h}^{(t-1)}$ to check whether the past memory is useful for the computation of the current memory.

$$f^{(t)} = \sigma(\mathbf{W}^{(f)}\mathbf{x}^{(t)} + \mathbf{U}^{(f)}\mathbf{h}^{(t-1)}) \tag{3.36}$$

4. **Final Memory Generation**. This stage takes the advice of the forget gate $f^{(t)}$ and, accordingly, forgets the past memory c^{t-1}. Similarly, it takes the advice of the input gate $i^{(t)}$ and, accordingly, gates the new memory $\hat{c}^{(t)}$. It then sums these two results to produce

the final memory $c^{(t)}$.

$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \hat{c}^{(t)} \tag{3.37}$$

5. **Output/Exposure Gate**. Its purpose is to separate the final memory from the hidden state. The final memory $c^{(t)}$ contains a lot of information that is not necessarily required to be saved in the hidden state. Hidden states are used in every single gate of an LSTM and thus, this gate makes the assessment regarding which parts of the memory $c^{(t)}$ need to be exposed/present in the hidden state $h^{(t)}$. The signal that it produces to indicate this is $o^{(t)}$ and it is used to gate the point-wise hyperbolic tangent of the memory.

$$o^{(t)} = \sigma(\mathbf{W}^{(0)}\mathbf{x}^{(t)} + \mathbf{U}^{(0)}\mathbf{h}^{(t-1)}) \tag{3.38}$$

$$h^{(t)} = o^{(t)} \circ \tanh(c^t) \tag{3.39}$$

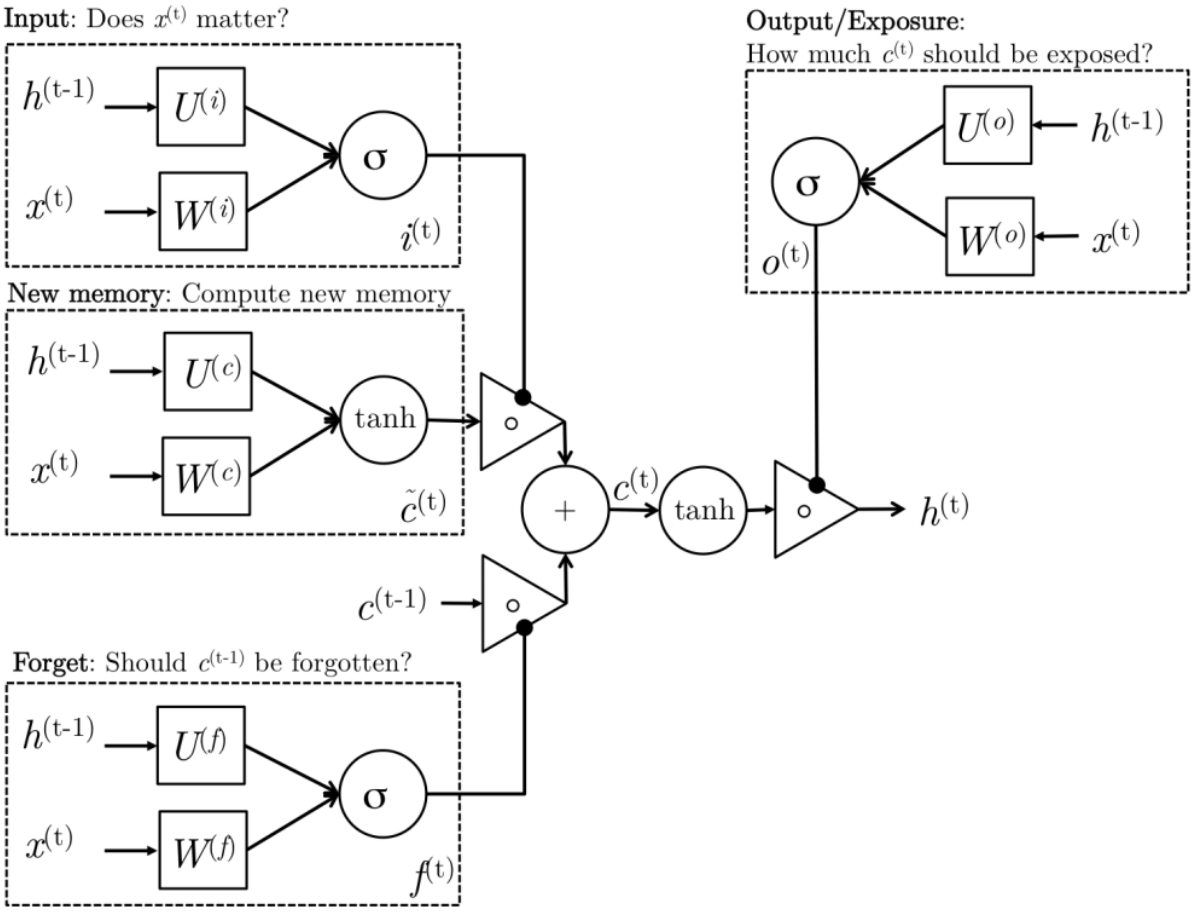


Figure 3.13: Long-Short-Term Memory cell structure [69].

3.4 Conclusion

This chapter has presented the necessary basic knowledge to understand both core network types that the LRCN utilises: ConvNets and LSTMs.

Firstly, the fundamentals of common NN were shown, specially fully-connected networks, as well as a review of the most important parameters to take into account when training NNs, such as output unit types, activation functions, the back-propagation algorithm and some regularisation and optimisation methods; with particular emphasis in those that are part of the LRCN: ReLUs, softmax output units, dropout regularisation and SGD method.

Secondly, the CNN architecture and all the possible hyperparameters that exist to configure it were described. Moreover, there is a brief description of the most common ConvNet architectures.

Finally, the chapter ends up with the theory about RNN, with special interest in LSTMs as a way to address the problem of long-term dependencies whose basic structure shown in figure 3.13 is the one used by the LRCN.

Chapter 4

Implementation: LRCN

Long-term Recurrent Convolutional Networks (LRCN) are a novel neural network architecture proposed by [1]. This architecture combines the demonstrated strengths of CNNs in visual recognition tasks with the fairly good performance of LSTMs in modelling time-dependent sequences. The LRCN structure is illustrated in figure 4.1.

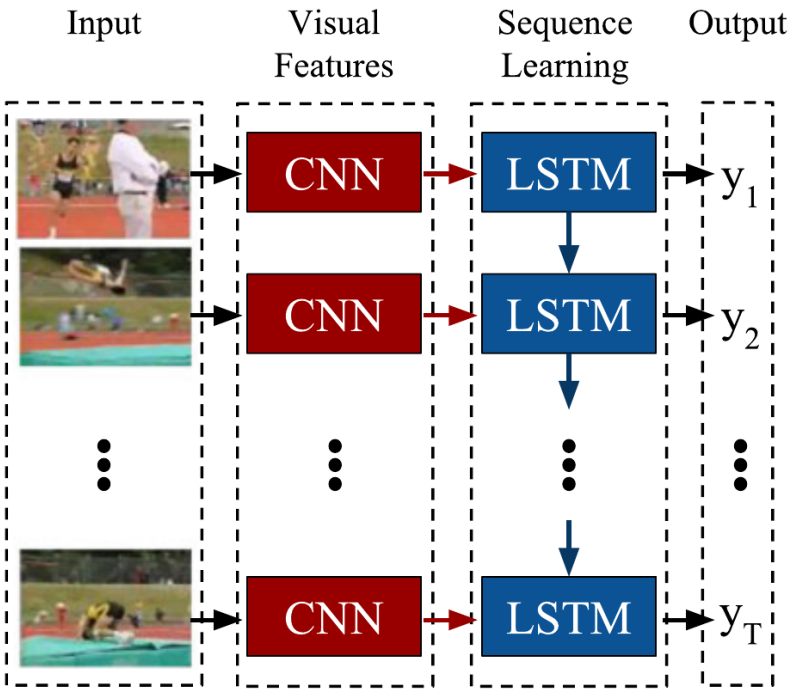


Figure 4.1: Long-term Recurrent Convolutional Network architecture [1].

As described in [1], this architecture has been designed to be used in three different visual tasks: activity recognition, image captioning and video description. Although the three tasks could be useful in the context of video surveillance, for the purpose of the present work this network is only evaluated in the **activity recognition** task.

Within a video sequence, actions or events are always performed in various frames. There-

fore, it is very likely that there is a time relationship between the information of those frames. On the one hand, in the first stage the system launches one ConvNet for each frame of the input video sequence for extracting visual features. On the other hand, the LSTM stage uses those features to learn and model the time relationship between them throughout the video sequence. At the output, the LRCN gives an estimate of the action that is being performed. It is worth noting that this architecture is end-to-end trainable. In other words, the gradient can be propagated backwards through all the network so all the parameters are trained.

The whole system has been implemented using the widely adopted deep learning framework *Caffe* [13], whose root code is written in C++ with CUDA [14] for GPU computation.

4.1 Visual Feature Extraction

The visual features utilised in the second stage by the stacked LSTMs are obtained from a CNN. This ConvNet is the ZFNet [65] with slight modifications in certain parameters of some layers. The structure of the model is shown in figure 4.2.

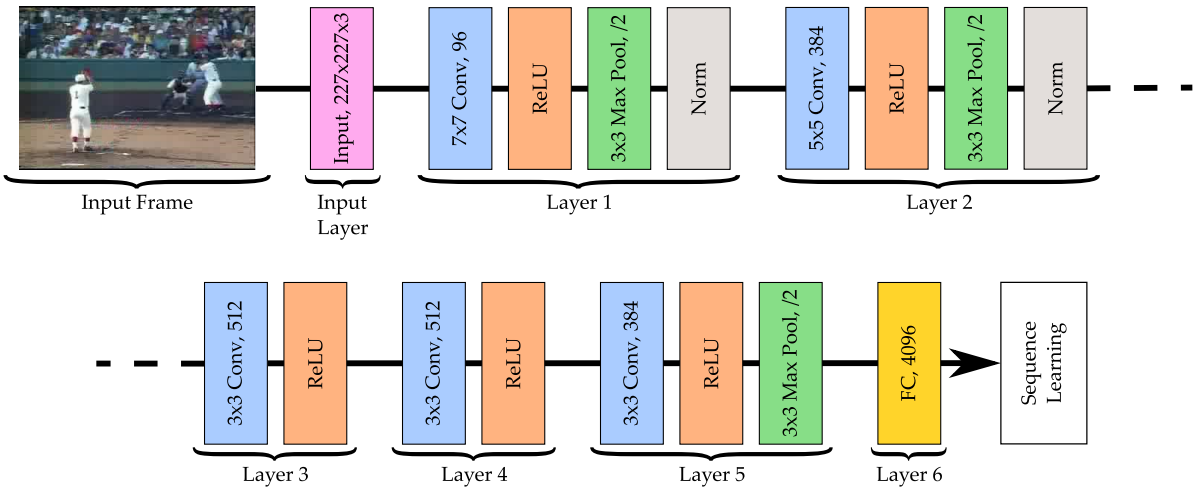


Figure 4.2: CNN used in the visual feature extraction stage.

As seen in figure 4.2, features are extracted with a 6 layer CNN model, the first five are convolutional and the last layer is fully-connected. The input layer is the one that adjusts input images to fit the dimensions. It resizes every image to 256×256, without keeping the aspect ratio, and then takes a random 227×227 crop. The only preprocessing applied to the resulting crop before being used as input is subtracting the mean value over the training set from each pixel. As the network works with RGB images, the input is 3-dimensional, corresponding each dimension to a colour channel. Finally, this volume is the input volume of the next layer.

The first convolutional layer takes the input volume of size 227×227×3 generated in the previous layer and convolves it with 96 different filters, each of size 7×7, using a stride of

2 and no zero-padding. According to the equation 3.24, the output volume is 111×111×96. This volume is followed by a rectified linear activation function (see section 3.1.3) and max pooling within 3×3 regions and a stride of 2. To enhance network generalisation, local response normalisation (described in [64]) is applied after the pooling. The dimension of the final volume is 55×55×96.

The second convolutional layer convolves the input volume with 384 filters of size 5×5, stride of 2 and no zero-padding, forming an output volume of 27×27×384. As for the first convolutional layer, this one is also followed by a rectified linear activation function, max pooling and local response normalisation, yielding a 13×13×384 volume.

The third and fourth layers work with 512 filters of size 3×3, stride of 1 and 1 pixel zero-padding. After each layer, rectified linear functions are applied to the outputs but neither pooling nor normalisation are used.

The fifth layer is also convolutional and uses 384 filters of same size, stride and zero-padding as in layers 3 and 4. After this layer, ReLU and max-pooling with 3×3 regions and stride of 2 are used again, so the final output is a 6×6×384 volume.

Afterwards, the last feature map is taken by a fully-connected layer to produce a 4096 feature vector. Finally, this feature vector is the input of the LSTM-based stage.

4.2 Sequence Learning and Action Prediction

Sequence learning is achieved through a recurrent module based on LSTM with the same structure as in figure 3.13. Each LSTM network has 1024 hidden units and all of them are connected as figure 4.3 below demonstrates.

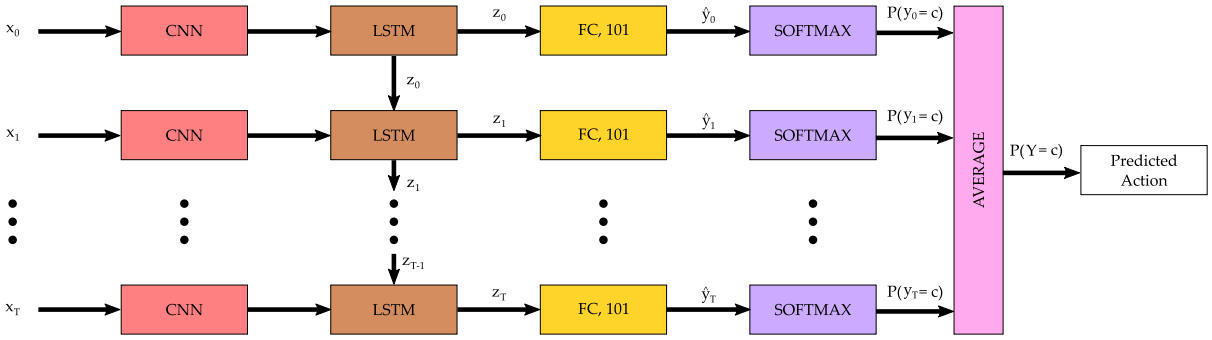


Figure 4.3: Sequence Learning and Action Prediction stage.

The output of the previous stage of the system is a fixed length vector corresponding to the visual features of each frame of a video sequence and each LSTM network of the recurrent module has this feature vector connected to its input gate. The LSTMs produce after their

last hidden unit an output vector z_t which is also connected to the memory gates of the next network. Therefore, this module results in a stack of LSTM networks atop one another and working with information of previous and current frames of the video sequence with intent to learn complex and long-term temporal dynamic of actions.

Regarding action prediction, in this system, the overall classification problem consists in obtaining a global action prediction for a video sequence by analysing each frame. In other words, the system must give a static output (prediction) based on a sequential input (frames). To work out this problem, action prediction is given by a simple two-step aproach.

Firstly, the system gives a prediction of the probability distribution for each frame outcome ($P(y_t)$) over all possible actions, with $y_t \in C$ (where C is the total number of frames) . To get this distribution, $z_t \in \mathbb{R}^{d_z}$ (d_z is the number of elements of z_t) outputs are passed through a linear fully-connected network which performs the biased inner product of equation 4.1, where $\mathbf{W}_z \in \mathbb{R}^{|C| \times d_z}$ and $\mathbf{b_z} \in \mathbb{R}^{|C|}$.

$$\mathbf{\hat{y}_t} = \mathbf{W}_z \mathbf{z_t} + \mathbf{b_z} \tag{4.1}$$

Then, the probability distribution is computed by taking the softmax of $\mathbf{\hat{y}_t}$ as in equation 4.2 where c is the total number of possible actions.

$$P(y_t = c) = softmax(\mathbf{\hat{y}_t}) = \frac{e^{\hat{y}_{t,c}}}{\sum_{c'} e^{\hat{y}_{t,c'}}} \tag{4.2}$$

In the second step, the average between all per-frame probabilities is taken, yielding a probability distribution for the whole video sequence. Therefore, the predicted action will correspond to the highest value of the average probability distribution.

As previously mentioned, this system is end-to-end trainable. This means that the parameters of the model's visual and sequential components can be jointly optimised. Optimisation is carried out by maximising the likelihood of the ground truth outputs y_t at each time step t, conditioned on the input data and labels up to that point $(x_{1:t}, y_{1:t-1})$. In particular, for a training set $\mathcal{D}$ of labelled sequences $(x_t, y_t)_{t=1}^{T} \in \mathcal{D}$, and being (V, W) the parameters of the system's visual and sequential stages, the loss function in 4.3 minimises the expected negative log-likelihood of a sequence sampled from the training set.

$$\mathcal{L}(V, W, \mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(x_t, y_t)_{t=1}^{T} \in \mathcal{D}} \sum_{t=1}^{T} \log P(y_t / x_{1:t}, y_{1:t-1}, V, W) \tag{4.3}$$

This way, the V parameters of the visual feature extractor learn to pick out those aspects of

the visual input that are most relevant to the sequence learning stage. Likewise, V parameters learn how best to interpret the dynamic of all considered actions.

4.3 Conclusion

The implementation of the Long-term Recurrent Convolutional Network architecture has been explained in this chapter. This network is a two-stage system. The first stage extracts visual features while the second stage uses those features to model the time relationship between them throughout the video sequence. The whole system has been implemented using the deep learning framework *Caffe* [13].

The visual feature extraction stage is based on the 6 layer CNN model shown in figure 4.2. This ConvNet is the ZFNet[65] with slight modifications of certain parameters of some layers. The first five layers are convolutional and the last one is fully-connected.

The sequence learning stage is a recurrent module based on LSTM. Each LSTM network of figure 4.3 has 1024 hidden units and works with the corresponding feature vector generated in the previous stage. The output of every LSTM is also connected to the memory gate of the next network so the module can work with information of previous and current frames of the video sequence.

The system gives a prediction of the action which is being performed in the sequence by averaging all the per-frame probability distributions. Those per-frame probability distributions are computed by taking the softmax of each output of the sequence learning stage.

It is worth stressing that the implemented system is end-to-end trainable. In other words, the parameters of the model's visual and sequential components can be jointly optimised.

Chapter 5

Results

This chapter contains the evaluation of the LRCN architecture. All the experiments have been performed using the UCF101 dataset [2], which consists of 101 action classes in over 13,000 clips collected from YouTube.

Furthermore, a baseline model based on the ZFNet [65] has been defined in order to check the improvement of this new proposed architecture in the action recognition task. Besides the baseline, the performance of the system has been also compared to other proposed methods.

5.1 Experimental Set-up

Before showing the results of the experiments, it is important to clarify the conditions in which thereof have been carried out. The dataset and baseline used are described here, as well as how the system has been trained.

5.1.1 UCF101 Dataset

The UCF101 [2] is an action recognition dataset which includes a total number of 101 action classes divided into five types: Human-Object Interaction, Body-Motion Only, Human-Human Interaction, Playing Musical Instruments and Sports.

The dataset is composed of 13,320 videos which have been recorded in unconstrained environments and typically include large variations in camera motion, different lighting conditions, partial occlusion and low quality frames, thus being a challenging dataset. Figure 5.1 shows sample frames of 6 action classes from UCF101.

Videos were collected from *YouTube* with a fixed frame rate of 25 FPS and a resolution of

Figure 5.1: Sample frames for 6 action classes of UCF101 [2].

320×240. Clips of one action class are divided into 25 groups which contain 4-7 clips. All the clips of one group share some common features, such as background, viewpoint or actors.

As opposed to other datasets such as Thumos'14 [70], Thumos'15 [71] or ActivityNet [72]; UCF101 videos are temporally trimmed, in other words, videos have been manually trimmed so they only contain the part of the video where the action is being performed.

5.1.2 Baseline Method

A baseline has been used in order to compare the improvements of adding a sequence learning method based on LSTM to a simple convolutional network structure. This baseline is, in essence, the CNN used in the first stage of the LRCN previously described in section 4.1. The difference between the one shown in figure 4.2 and the baseline is that the latter has two more fully-connected layers and the softmax operation afterwards. This structure is illustrated in figure 5.2.

In a video, all frames are individually classified by the baseline presented. As in the LRCN architecture, the whole video is classified by averaging all per-frame probabilities. The main difference is that the baseline only works with information of the current frame, unlike the LRCN model which works with information of previous and current frames.

5.1.3 Training the Model

The LRCN model is trained and tested with the UCF101 dataset [2]. Both RGB frames and optical flow are considered as inputs to the system, using [73] to compute optical flow. When

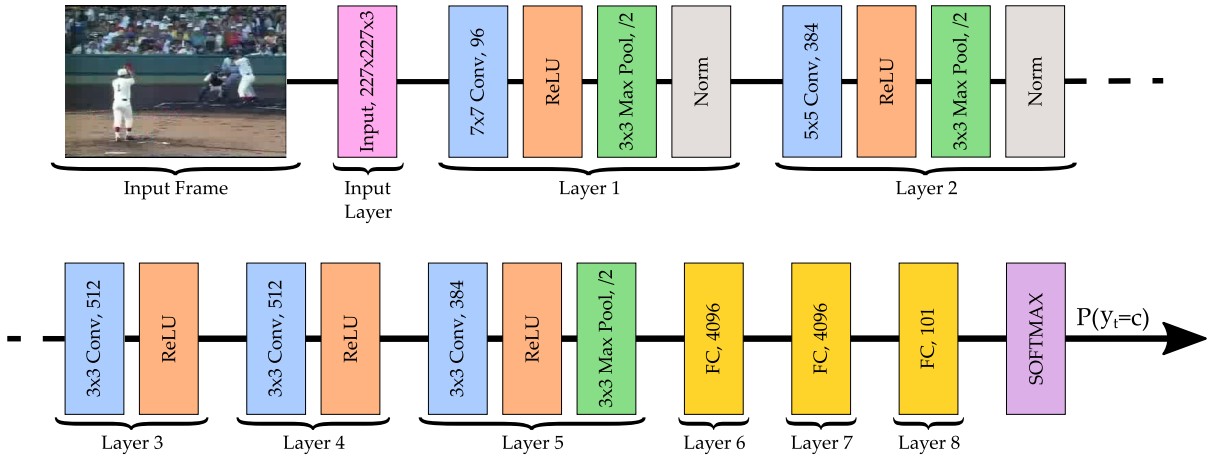


Figure 5.2: Baseline used to compare the LRCN with.

optical flow is used as input, the first and second channel correspond to the x and y flow values, respectively, scaled to a range of $[-128, +128]$. The third channel represents the flow magnitude.

The UCF101 is divided into a training set with over 8,000 videos and a test set containing the rest. In turn, the training set is split into three splits. To complete a thorough training but avoiding overfitting, the system is trained with cross-validation over the three splits. Additionally, another way to avoid overfitting is augmenting the data. This is made by mirroring the 227×227 random crops.

It is worth recalling that the whole LRCN system is end-to-end trainable, which means that it is not necessary to train both stages separately. However, this fact does not imply that one of the stages can be pre-trained. Actually, the CNN used in the visual feature extraction stage is pre-trained on the classification training set of the ImageNet [74] dataset. This previous initialisation of weights makes the whole training faster and helps to avoid overfitting. Regarding the sequence learning stage, when using RGB as input, each LSTM has 256 hidden units. In contrast, with optical flow as input, 1024 hidden units are used.

To help the whole system to generalise, dropout is applied to the last fully-connected layer of the system with a ratio of 0.5, meaning that in each iteration every neuron of that layer has a probability of 0.5 of remaining active.

Both stages of the LRCN architecture are jointly trained using SGD with momentum of value 0.9, and the hardware utilised to train the network is an *NVIDIA GeForce GTX 1080* [10].

5.2 Evaluation

The evaluation of the trained LRCN system is performed on the test set of the UCF101. As with the training set, the test set is also split into three splits and the performance of the system is measured by the mean classification accuracy across the three splits.

The performance of the network is measured when using RGB and flow as inputs. Furthermore, like proposed in [4], the RGB and flow trained systems can be combined by computing a weighted average of the softmax scores.

Table 5.1 shows the average accuracy of the baseline and the LRCN system across the three splits of the UCF101 test set. It can be concluded that in every case, the proposed model outperforms the baseline. Results also prove that using optical flow as input increases the accuracy of the system. Regarding to the combination of RGB and flow networks, using a higher weight for the flow scores results in a better accuracy. The best accuracy value is obtained when fusing RGB and flow LRCN systems with weights $1/3$ and $2/3$, respectively.

	Single Network		**Weighted Average**	
	RGB	**Flow**	**1/2, 1/2**	**1/3, 2/3**
Baseline Model	67.37	74.37	75.46	78.94
LRCN Architecture	68.20	78.47	81.56	84.12

Table 5.1: Average accuracy of the baseline and the LRCN system across the three splits of the UCF101 test set, with RGB and flow as inputs.

Table 5.2 contains the performance of other state-of-the-art methods on the UCF101 dataset. One of them is a hand-crafted method based on Improved Dense Trajectories (IDT) and the rest are deep learning based models that also fuse RGB and flow networks. It can be noticed that although [6] is a classic proposal, it achieves 89.1% of accuracy. The best system is the work proposed by [8] with an average accuracy of 91.7%. Therefore, the LRCN achieves a comparable average accuracy value to other state-of-the-art models.

Method	**Accuracy**
Slow Fusion [3]	65.4%
Spatial-Stream [4]	73.0%
LRCN Architecture	84.12%
Two-Stream [4]	86.9%
CNN + LSTM [5]	88.6
IDT [6]	89,1%
C3D[7]	90.4%
LTC [8]	91.7%

Table 5.2: Average accuracy of state-of-the-art methods. LRCN architecture shows a performance comparable to the state of the art. The information within this table has been obtained from [8] and [4]

5.3 Conclusion

This chapter has shown the performance of the LRCN system. This system has been trained and tested with the UCF101 dataset [2], with both RGB and optical flow as inputs. Since the network is end-to-end trainable, both stages have been jointly trained using SGD with 0.9 of momentum and 0.5 dropout after the last fully-connected layer. To avoid overfitting and get the best generalisation of the system, cross-validation over the three splits of the UCF101 training set has been used.

The performance of the system has been measured by computing the average classification accuracy across the three splits of the UCF101 test set, yielding a 68.2% and 78.47% when using RGB and flow as inputs, respectively. The best performance is obtained when fusing RGB and flow trained systems by averaging the softmax scores of each one. In this case the average accuracy is 84.12%. In any way, the LRCN outperforms the baseline model. These results are reported in table 5.1.

Regarding the comparative with other state-of-the-art methods shown in 5.2, the implemented system achieves a comparable average accuracy value to that of other proposed models.

Chapter 6

Conclusion and Future Work

After completing this work, this chapter shows some conclusions about the network that has been implemented and evaluated, as well as several future directions to develop a whole anomaly detection system.

6.1 Conclusion

The master's thesis here presented has consisted of implementing and evaluating a state-of-the-art action recognition method suitable for an anomaly detection system. The method is the LRCN shown in figure 4.1 which is a two-stage configuration based on CNN and LSTM.

Prior to the analysis of the developed method, a study of the recent literature about anomaly detection has been made so as to determine the route the scientific community is following. This study suggest that, although there are some recent works with semi-supervised anomaly detection systems, unsupervised ones are the most popular. Furthermore, in spite of the fact that trajectory-based and low-level feature methods have traditionally been used, a new line of research based on deep learning architectures is beginning to be widely explore.

Regarding the implementation of the LRCN, the visual feature extraction stage utilises a 6 layer CNN model while the sequence learning stage works with LSTM networks. Due to this structure, the time-relationship between the visual features can be modelled.

The LRCN is end-to-end trainable, therefore its two stages are jointly trained using SGD with momentum and considering RGB and optical flow as inputs. For both training and testing, the UCF101 dataset [2] has been used.

The results of the experiments suggest that the performance of the LRCN is better than that of the baseline for both RGB and flow inputs, concluding that the idea of learning the temporal

dynamic of the visual features is beneficial. Particularly, the best results are obtained when fusing RGB and flow trained systems, achieving a 84.12% of classification accuracy.

When comparing with other state-of-the-art alternatives, the implemented system obtains a comparable accuracy value.

6.2 Future Work

Since the global would be to implement a complete anomaly detection system, several future directions are proposed here. The idea would be to go from a supervised or semi-supervised system to an unsupervised one.

- **Explore other datasets**. To improve the LRCN generalisation, other datasets with different action classes could be used.
- **Semi-supervised anomaly detection.** Having the action recognition implemented and trained to detect a certain number of classes, an anomaly would be those actions appearing in a video which do not correspond to any class. This approach is semi-supervised since the anomaly label is not considered during training, and additionally, it would be the very next step of the work here presented.
- **Unsupervised anomaly detection.** In order to get an unsupervised system, there are several clustering based approaches that can be explored. The core of an unsupervised system could be the LRCN trained for action detection but without the softmax output. Any clustering algorithm would be applied to the features of the last fully-connected network to find some pattern that separates known action features from unknown situation features.
- **Similar architectures.** Should it be required to use another core network in the anomaly detection system, the work has demonstrate that it is preferable to use an architecture capable of learning time relationships between visual features from continuous frames, as the LRCN does.

References

[1] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, T. Darrell, and K. Saenko, "Long-term recurrent convolutional networks for visual recognition and description", in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Institute of Electrical and Electronics Engineers (IEEE), Jun. 2015. DOI: 10.1109/cvpr.2015.7298878.

[2] K. Soomro, A. R. Zamir, and M. Shah, "UCF101: A dataset of 101 human actions classes from videos in the wild", *CoRR*, vol. abs/1212.0402, 2012. [Online]. Available: http://arxiv.org/abs/1212.0402.

[3] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks", in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, IEEE, Jun. 2014. DOI: 10.1109/cvpr.2014.223.

[4] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos", in *Proceedings of the 27th International Conference on Neural Information Processing Systems*, ser. NIPS'14, Montreal, Canada: MIT Press, 2014, pp. 568–576. [Online]. Available: http://dl.acm.org/citation.cfm?id=2968826.2968890.

[5] J. Y.-H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: Deep networks for video classification", in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2015. DOI: 10.1109/cvpr.2015.7299101.

[6] Z. Lan, M. Lin, X. Li, A. G. Hauptmann, and B. Raj, "Beyond gaussian pyramid: Multi-skip feature stacking for action recognition", in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2015. DOI: 10.1109/cvpr.2015.7298616.

[7] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks", in *2015 IEEE International Conference on Computer Vision (ICCV)*, IEEE, Dec. 2015. DOI: 10.1109/iccv.2015.510.

[8] G. Varol, I. Laptev, and C. Schmid, "Long-term Temporal Convolutions for Action Recognition", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[9] V. Saligrama and Z. Chen, "Video anomaly detection based on local statistical aggregates", in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, Institute of Electrical and Electronics Engineers (IEEE), Jun. 2012. DOI: 10.1109/CVPR.2012.6247917.

[10] NVIDIA. (2017). Nvidia web site. NVIDIA, Ed., [Online]. Available: http://www.nvidia.co.uk/page/home.html.

[11] M. W. Green, "The appropriate and effective use of security technologies in u.s. schools", National Institute of Justice, research rep., Sep. 1999. DOI: DOI:10.2172/974410.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[13] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe", in *Proceedings of the ACM International Conference on Multimedia - MM '14*, Association for Computing Machinery (ACM), 2014. DOI: 10.1145/2647868.2654889.

[14] NVIDIA. (2017). Cuda parallel computing. NVIDIA, Ed., [Online]. Available: http://www.nvidia.co.uk/object/cuda-parallel-computing-uk.html.

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors", *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986. DOI: 10.1038/323533a0.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[17] F. Nater, "Abnormal behavior detection in surveillance videos", PhD thesis, ETH, 2012. DOI: 10.3929/ethz-a-007309397.

[18] M. Markou and S. Singh, "Novelty detection: A review—part 1: Statistical approaches", *Signal Processing*, vol. 83, no. 12, pp. 2481–2497, Dec. 2003. DOI: 10.1016/j.sigpro.2003.07.018.

[19] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey", *ACM Computing Surveys*, vol. 41, no. 3, pp. 1–58, Jul. 2009. DOI: 10.1145/1541880.1541882. [Online]. Available: https://doi.org/10.1145%2F1541880.1541882.

[20] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, "A review of novelty detection", *Signal Processing*, vol. 99, pp. 215–249, Jun. 2014. DOI: 10.1016/j.sigpro.2013.12.026.

[21] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data", *PLOS ONE*, vol. 11, no. 4, D. Zhu, Ed., e0152173, Apr. 2016. DOI: 10.1371/journal.pone.0152173.

[22] Y. Wang, K. Huang, and T. Tan, "Abnormal activity recognition in office based on r transform", in *2007 IEEE International Conference on Image Processing*, Institute of Electrical and Electronics Engineers (IEEE), Sep. 2007. DOI: 10.1109/icip.2007.4378961. [Online]. Available: https://doi.org/10.1109%2Ficip.2007.4378961.

[23] T. Duong, H. Bui, D. Phung, and S. Venkatesh, "Activity recognition and abnormality detection with the switching hidden semi-Markov model", in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, Institute of Electrical and Electronics Engineers (IEEE), 2005. DOI: 10.1109/cvpr.2005.61. [Online]. Available: https://doi.org/10.1109%2Fcvpr.2005.61.

[24] H. Dee and D. Hogg, "Detecting inexplicable behaviour", in *Procedings of the British Machine Vision Conference 2004*, British Machine Vision Association and Society for Pattern Recognition, 2004. DOI: 10.5244/c.18.50. [Online]. Available: https://doi.org/10.5244%2Fc.18.50.

[25] O. Boiman and M. Irani, "Detecting irregularities in images and in video", *International Journal of Computer Vision*, vol. 74, no. 1, pp. 17–31, Jan. 2007. DOI: 10.1007/s11263-006-0009-9. [Online]. Available: https://doi.org/10.1007%2Fs11263-006-0009-9.

[26] R. Mehran, A. Oyama, and M. Shah, "Abnormal crowd behavior detection using social force model", in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Institute of Electrical and Electronics Engineers (IEEE), Jun. 2009. DOI: 10.1109/cvpr.2009.5206641.

[27] Y. Benezeth, P.-M. Jodoin, V. Saligrama, and C. Rosenberger, "Abnormal events detection based on spatio-temporal co-occurences", in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Institute of Electrical and Electronics Engineers (IEEE), Jun. 2009. DOI: 10.1109/cvpr.2009.5206686.

[28] A. Zaharescu and R. Wildes, "Anomalous behaviour detection using spatiotemporal oriented energies, subset inclusion histogram comparison and event-driven processing", in *Computer Vision – ECCV 2010*, Springer Nature, 2010, pp. 563–576. DOI: 10.1007/978-3-642-15549-9_41.

[29] V. Mahadevan, W. Li, V. Bhalodia, and N. Vasconcelos, "Anomaly detection in crowded scenes", in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Institute of Electrical and Electronics Engineers (IEEE), Jun. 2010. DOI: 10.1109/cvpr.2010.5539872.

[30] L. Kratz and K. Nishino, "Anomaly detection in extremely crowded scenes using spatio-temporal motion pattern models", in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Institute of Electrical and Electronics Engineers (IEEE), Jun. 2009. DOI: 10.1109/cvpr.2009.5206771.

[31] F. Tung, J. S. Zelek, and D. A. Clausi, "Goal-based trajectory analysis for unusual behaviour detection in intelligent surveillance", *Image and Vision Computing*, vol. 29, no. 4, pp. 230–240, Mar. 2011. DOI: 10.1016/j.imavis.2010.11.003.

[32] D. Zhang, D. Gatica-Perez, S. Bengio, and I. McCowan, "Semi-supervised adapted HMMs for unusual event detection", in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, Institute of Electrical and Electronics Engineers (IEEE), 2005. DOI: 10.1109/cvpr.2005.316. [Online]. Available: https://doi.org/10.1109%2Fcvpr.2005.316.

[33] F. Nater, H. Grabner, T. Jaeggli, and L. van Gool, "Tracker trees for unusual event detection", in *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, Institute of Electrical and Electronics Engineers (IEEE), Sep. 2009. DOI: 10.1109/iccvw.2009.5457578. [Online]. Available: https://doi.org/10.1109%2Ficcvw.2009.5457578.

[34] Y. Cong, J. Yuan, and Y. Tang, "Video anomaly search in crowded scenes via spatio-temporal motion context", *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 10, pp. 1590–1599, Oct. 2013. DOI: 10.1109/tifs.2013.2272243.

[35] C. Lu, J. Shi, and J. Jia, "Abnormal event detection at 150 FPS in MATLAB", in *2013 IEEE International Conference on Computer Vision*, Institute of Electrical and Electronics Engineers (IEEE), Dec. 2013. DOI: 10.1109/iccv.2013.338.

[36] Y. Cong, J. Yuan, and J. Liu, "Sparse reconstruction cost for abnormal event detection", in *CVPR 2011*, Institute of Electrical and Electronics Engineers (IEEE), Jun. 2011. DOI: 10.1109/cvpr.2011.5995434.

[37] R. Sillito and R. Fisher, "Semi-supervised learning for anomalous trajectory detection", in *Procedings of the British Machine Vision Conference 2008*, British Machine Vision Association and Society for Pattern Recognition, 2008. DOI: 10.5244/c.22.103.

[38] B. Antic and B. Ommer, "Video parsing for abnormality detection", in *2011 International Conference on Computer Vision*, Institute of Electrical and Electronics Engineers (IEEE), Nov. 2011. DOI: 10.1109/iccv.2011.6126525.

[39] F. Jiang, J. Yuan, S. A. Tsaftaris, and A. K. Katsaggelos, "Anomalous video event detection using spatiotemporal context", *Computer Vision and Image Understanding*, vol. 115, no. 3, pp. 323–333, Mar. 2011. DOI: 10.1016/j.cviu.2010.10.008. [Online]. Available: https://doi.org/10.1016%2Fj.cviu.2010.10.008.

[40] S. Calderara, U. Heinemann, A. Prati, R. Cucchiara, and N. Tishby, "Detecting anomalies in people's trajectories using spectral graph analysis", *Computer Vision and Image Understanding*, vol. 115, no. 8, pp. 1099–1111, Aug. 2011. DOI: 10.1016/j.cviu.2011.03.003.

[41] J. Kim and K. Grauman, "Observe locally, infer globally: A space-time MRF for detecting abnormal activities with incremental updates", in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Institute of Electrical and Electronics Engineers (IEEE), Jun. 2009. DOI: 10.1109/cvpr.2009.5206569.

[42] C. Piciarelli and G. Foresti, "On-line trajectory clustering for anomalous events detection", *Pattern Recognition Letters*, vol. 27, no. 15, pp. 1835–1842, Nov. 2006. DOI: 10.1016/j.patrec.2006.02.004.

[43] B. T. Morris and M. M. Trivedi, "Trajectory learning for activity understanding: Unsupervised, multilevel, and long-term adaptive approach", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 11, pp. 2287–2301, Nov. 2011. DOI: 10.1109/tpami.2011.64.

[44] C. Piciarelli, C. Micheloni, and G. Foresti, "Trajectory-based anomalous event detection", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 11, pp. 1544–1554, Nov. 2008. DOI: 10.1109/tcsvt.2008.2005599.

[45] D. Xu, R. Song, X. Wu, N. Li, W. Feng, and H. Qian, "Video anomaly detection based on a hierarchical activity discovery within spatio-temporal contexts", *Neurocomputing*, vol. 143, pp. 144–152, Nov. 2014. DOI: 10.1016/j.neucom.2014.06.011.

[46] H. Mousavi, M. Nabi, H. K. Galoogahi, A. Perina, and V. Murino, "Abnormality detection with improved histogram of oriented tracklets", in *Image Analysis and Processing — ICIAP 2015*, Springer Nature, 2015, pp. 722–732. DOI: 10.1007/978-3-319-23234-8_66.

[47] M. J. Roshtkhari and M. D. Levine, "An on-line, real-time learning method for detecting anomalies in videos using spatio-temporal compositions", *Computer Vision and Image Understanding*, vol. 117, no. 10, pp. 1436–1452, Oct. 2013. DOI: 10.1016/j.cviu.2013.06.007.

[48] ——, "Online dominant and anomalous behavior detection in videos", in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, Institute of Electrical and Electronics Engineers (IEEE), Jun. 2013. DOI: 10.1109/cvpr.2013.337.

[49] S. Wu, B. E. Moore, and M. Shah, "Chaotic invariants of lagrangian particle trajectories for anomaly detection in crowded scenes", in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Institute of Electrical and Electronics Engineers (IEEE), Jun. 2010. DOI: 10.1109/cvpr.2010.5539882.

[50] T. Xiao, C. Zhang, and H. Zha, "Learning to detect anomalies in surveillance video", *IEEE Signal Processing Letters*, vol. 22, no. 9, pp. 1477–1481, Sep. 2015. DOI: 10.1109/lsp.2015.2410031.

[51] M. Bertini, A. D. Bimbo, and L. Seidenari, "Multi-scale and real-time non-parametric approach for anomaly detection and localization", *Computer Vision and Image Understanding*, vol. 116, no. 3, pp. 320–329, Mar. 2012. DOI: 10.1016/j.cviu.2011.09.009.

[52] M. Sabokrou, M. Fathy, M. Hoseini, and R. Klette, "Real-time anomaly detection and localization in crowded scenes", in *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Institute of Electrical and Electronics Engineers (IEEE), Jun. 2015. DOI: 10.1109/cvprw.2015.7301284.

[53] P. Antonakaki, D. Kosmopoulos, and S. J. Perantonis, "Detecting abnormal human behaviour using multiple cameras", *Signal Processing*, vol. 89, no. 9, pp. 1723–1738, Sep. 2009. DOI: 10.1016/j.sigpro.2009.03.016.

[54] W. Li, V. Mahadevan, and N. Vasconcelos, "Anomaly detection and localization in crowded scenes", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 1, pp. 18–32, Jan. 2014. DOI: 10.1109/tpami.2013.111.

[55] A. Adam, E. Rivlin, I. Shimshoni, and D. Reinitz, "Robust real-time unusual event detection using multiple fixed-location monitors", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 3, pp. 555–560, Mar. 2008. DOI: 10.1109/tpami.2007.70825.

[56] Y. Yuan, J. Fang, and Q. Wang, "Online anomaly detection in crowd scenes via structure analysis", *IEEE Transactions on Cybernetics*, vol. 45, no. 3, pp. 548–561, Mar. 2015. DOI: 10.1109/tcyb.2014.2330853.

[57] D. Xu, E. Ricci, Y. Yan, J. Song, and N. Sebe, "Learning deep representations of appearance and motion for anomalous event detection", in *Procedings of the British Machine Vision Conference 2015*, British Machine Vision Association and Society for Pattern Recognition, 2015. DOI: 10.5244/c.29.8.

[58] M. Ravanbakhsh, M. Nabi, H. Mousavi, E. Sangineto, and N. Sebe, "Plug-and-play cnn for crowd motion analysis: An application in abnormal event detection", *CoRR*, Oct. 2, 2016. arXiv: 1610.00307v1 [cs.CV].

[59] M. Sabokrou, M. Fayyaz, M. Fathy, and R. klette, "Fully convolutional neural network for fast anomaly detection in crowded scenes", *CoRR*, Sep. 3, 2016. arXiv: 1609.00866v1 [cs.CV].

[60] M. Sabokrou, M. Fayyaz, M. Fathy, and R. Klette, "Deep-cascade: Cascading 3d deep neural networks for fast anomaly detection and localization in crowded scenes", *IEEE Transactions on Image Processing*, pp. 1–1, 2017. DOI: 10.1109/tip.2017.2670780.

[61] Y. S. Chong and Y. H. Tay, "Abnormal event detection in videos using spatiotemporal autoencoder", *CoRR*, Jan. 6, 2017. arXiv: 1701.01546v1 [cs.CV].

[62] S. University. (2017). Cs231n: Convolutional neural networks for visual recognition, [Online]. Available: http://cs231n.github.io/.

[63] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. DOI: 10.1109/5.726791.

[64] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc.,

2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[65] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks", in *Computer Vision – ECCV 2014*, Springer Nature, 2014, pp. 818–833. DOI: 10.1007/978-3-319-10590-1_53.

[66] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions", in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Institute of Electrical and Electronics Engineers (IEEE), Jun. 2015. DOI: 10.1109/cvpr.2015.7298594.

[67] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition", *CoRR*, vol. abs/1409.1556, 2014.

[68] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Institute of Electrical and Electronics Engineers (IEEE), Jun. 2016. DOI: 10.1109/cvpr.2016.90.

[69] S. University. (2017). Cs224d: Deep learning for natural language processing, [Online]. Available: http://cs224d.stanford.edu/syllabus.html.

[70] Y.-G. Jiang, J. Liu, A. Roshan Zamir, G. Toderici, I. Laptev, M. Shah, and R. Sukthankar, *THUMOS challenge: Action recognition with a large number of classes*, http://crcv.ucf.edu/THUMOS14/, 2014.

[71] A. Gorban, H. Idrees, Y.-G. Jiang, A. Roshan Zamir, I. Laptev, M. Shah, and R. Sukthankar, *THUMOS challenge: Action recognition with a large number of classes*, http://www.thumos.info/, 2015.

[72] F. C. Heilbron, V. Escorcia, B. Ghanem, and J. C. Niebles, "ActivityNet: A large-scale video benchmark for human activity understanding", in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2015. DOI: 10.1109/cvpr.2015.7298698.

[73] T. Brox, A. Bruhn, N. Papenberg, and J. Weickert, "High accuracy optical flow estimation based on a theory for warping", in *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2004, pp. 25–36. DOI: 10.1007/978-3-540-24673-2_3.

[74] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge", *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá